



Zadání diplomové práce

Název:	MakeAPI
Student:	Bc. Ondřej Vaniš
Vedoucí:	Ing. David Bernhauer, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Mnoho restaurací, obchodů a institucí zveřejňuje data ve formě jednoduchých HTML stránek (např. denní menu). Taková data by mohla být využita pro další strojové zpracování nebo tvorbu mashup aplikací, ale překážkou jsou často problémy s CORS nebo formátem dat. Vytvořte open-source webovou aplikaci, která uživateli umožní vytvořit ze statických HTML stránek jednoduché API dle jeho pokynů tak, aby bylo možné k datům přistupovat právě pomocí tohoto API.

1. Analyzujte základní technologie a podobné komerční aplikace (crawling a jiné).
2. Na základě analýzy definujte požadavky na vaši aplikaci a jejich důležitost.
3. Navrhněte UI pro transformaci formátu HTML do JSON, příp. jiného vhodného formátu.
4. Proveďte softwarovou analýzu, návrh a naimplementujte základ aplikace, která bude pokrývat alespoň ty nejdůležitější definované požadavky.
5. Při implementaci se zaměřte především na:
 - modularitu (rozšiřitelnost),
 - zpracování chyb (např. změnou formátu),
 - autentizaci a autorizaci (na straně webových stránek i vaší aplikace),
 - cachování,
 - CORS.
6. Aplikaci vhodně otestujte.



**FAKULTA
INFORMAČNÍCH
TECHNOLÓGIÍ
ČVUT V PRAZE**

Diplomová práce

MakeAPI

Ondřej Vaniš

Katedra softwarového inženýrství

Vedoucí práce: Ing. David Bernhauer, Ph.D.

11. ledna 2024

Poděkování

Chtěl bych předně poděkovat svému vedoucímu, Ing. Davidu Bernhauerovi, Ph.D., za podporu a vedení mé diplomové práce. Poděkovat bych chtěl také Dr.-Ing. Martinu Novotnému, své rodině a nejbližším přátelům za jejich podporu v průběhu celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mé práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 11. ledna 2024

.....

České vysoké učení technické v Praze

Fakulta informačních technologií

© 2024 Ondřej Vaniš. Všechna práva vyhrazena.

Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Vaniš, Ondřej. *MakeAPI*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Abstrakt

Tato práce se zaměřuje na návrh a implementaci prototypu webové aplikace MakeAPI, která umožňuje uživatelům transformovat statické HTML webové stránky do JSON dokumentu. Transformovaná data zpřístupňuje skrze API endpoint. Aplikace je implementována v jazyce TypeScript s použitím frameworku Next.js. Výsledkem je funkční prototyp, který byl lokálně nasazen a manuálně i automatizovaně otestován. Byla ověřena jeho základní funkčnost. Součástí výsledku jsou popsány i další možnosti a náměty na další rozšíření aplikace.

Klíčová slova webová aplikace, HTML, JSON, API, CORS, HTTP cache, open-source, Next.js, scraping, transformace dat

Abstract

This thesis focuses on the design and implementation of a web application prototype, MakeAPI, which enables users to transform static HTML web pages into JSON document. The transformed data is made accessible through an API endpoint. The application is implemented in TypeScript language and uses the Next.js framework. The result is a functional prototype that has been deployed locally and tested both manually and automatically. Its basic functionality has been verified. The results also include a description of further possibilities and suggestions for the application's future expansion.

Keywords web application, HTML, JSON, API, CORS, HTTP cache, open-source, Next.js, scraping, data transformation

Obsah

Úvod	1
1 Teorie webu	3
1.1 Web a základní technologie	3
1.2 Vlastnosti HTTP	4
1.3 Proxy	7
1.4 HTTP Cache	8
1.5 Rate-limiting	10
1.6 CORS protokol	11
1.7 Crawling a scraping	12
2 Rešerše	15
2.1 Analýza nástrojů pro scraping webových dat	15
2.2 Shrnutí analyzovaných nástrojů	21
3 Analýza	29
3.1 Popis projektu	29
3.2 Analýza požadavků	30
3.3 Modelování případů užití	33
3.4 Modelování procesů	43
4 Návrh	45
4.1 Použité technologie	45
4.2 Architektura aplikace	46
4.3 Diagram nasazení	48
4.4 Databázový model	49
4.5 Transformace dat	50
4.6 Rate-limiting	54
5 Implementace prototypu	57

5.1	Zdrojový kód	57
5.2	Struktura aplikace MakeAPI	57
5.3	Služby aplikace	61
5.4	Uživatelské rozhraní	68
5.5	URL Fetcher	69
6	Testování	71
6.1	Testovací prostředí	71
6.2	Manuální testy	73
6.3	Automatizované testy	83
6.4	Doporučení na základě testů	85
6.5	Návrhy na budoucí testování	85
7	Budoucí práce	87
7.1	Nasazení na produkční prostředí	87
7.2	Rozšíření možností autentizace	87
7.3	Využití umělé inteligence	88
7.4	Rozšíření možností transformace	88
7.5	Interakce se stránkou	91
7.6	Crawling	91
7.7	Rozšíření uživatelského rozhraní	92
7.8	Rozšíření možností exportu a validace	93
	Závěr	95
	Bibliografie	97
A	Seznam použitých zkratk	101
B	Ukázky UI aplikace	103
C	Obsah příloženého CD	111

Seznam obrázků

1.1	Model komunikace klient/server	3
1.2	Forward-proxy a reverse-proxy	7
2.1	Request Builder ve službě ScrapingBee	24
2.2	Dashboard aktéru ve službě Apify	25
2.3	Ukázka extrakce v aplikaci Diffbot	25
2.4	Ukázka specifikace vlastních pravidel v rámci <i>Custom API</i> ve službě Diffbot	26
2.5	Ukázka extrakce datových sloupců v aplikaci <i>Import.io</i>	26
2.6	Ukázka vytváření akcí ve službě <i>Octoparse</i>	27
2.7	Ukázka tutoriálu ve službě <i>ParseHub</i>	27
2.8	Ukázka struktury extrahovaných dat <i>ParseHub</i>	28
2.9	Ukázka výběru dat ve službě <i>Portia</i>	28
3.1	Diagram aktérů	34
3.2	Přehled vazeb mezi use-case a aktéry	41
3.3	Proces vytvoření transformace a přístupu skrze API endpoint	43
3.4	Proces zablokování transformace při přístupu skrze API	44
4.1	Komponenty MakeAPI aplikace	47
4.2	Diagram nasazení aplikace MakeAPI	49
4.3	Databázový model aplikace	51
4.4	Reprezentace transformačního schématu s využitím composite patternu	52
4.5	UI komponenty transformace	53
4.6	TransformService s <code>context</code> objektem pro zaznamenávání chyb během transformace	54
4.7	RateLimiter objekt pro zaznamenávání přístupů	55
5.1	Struktura URL stránek aplikace MakeAPI	60
5.2	Transformační editor	68

6.1	robots.txt umožňuje přístup ke zdroji	74
6.2	robots.txt se nepodařilo načíst	75
6.3	robots.txt zakazuje přístup ke zdroji	75
6.4	Nepodařilo se načíst dokument na zadané URL	76
6.5	Neočekávaný Content-Type	76
6.6	Nastavení testu pro ověření hlaviček odeslaných z URL Fetcheru	77
6.7	Ukázka komunikace mezi kontejnery v testovacím prostředí	79
6.8	Chyba při načítání obsahu z API endpointu	81
6.9	Úspěšné načtení obsahu z API endpointu	82
6.10	E-mailová notifikace o zablokování další extrakce	82
6.11	Výsledek end-to-end testů	84
7.1	Využití ChatGPT pro definici transformačního schématu	89
7.2	Příklad možného rozšíření o šablony/podschémata	91
7.3	Příklad interaktivního výběru prvků pomocí kurzoru	92
B.1	Úvodní stránka	104
B.2	Stránka projektu	105
B.3	Transformační editor	106
B.4	Formulář vytvoření dokumentu v projektu	107
B.5	Formulář pro přidání access-tokenu	108
B.6	Detail profilu uživatele	109
B.7	Možnost smazání účtu	109
B.8	Administrace uživatelů	110
B.9	Nastavení rate-limitů	110

Seznam tabulek

2.1	Shrnutí analýzy nástrojů pro scrapování dat	23
3.1	Pokrytí funkčních požadavků případy užití	42
4.1	Tabulka typů uzlů a jejich příslušná komponenta ve formuláři . . .	52
5.1	URL Fetcher API	69
6.1	Přehled dostupných služeb/kontejnerů	73

Seznam výpisů kódu

1.1	Ukázka URL	4
1.2	Ukázka HTTP requestu (vlevo) a HTTP response (vpravo) . . .	5
1.3	Ukázka souboru robots.txt, který zakazuje přístup k adresáři /admin všem robotům.	13
2.1	Ukázka definice <i>extract rules</i> pro extrakci dat z blogu ve ScrapingBee	16
2.2	Ukázka konfigurace Web Scraperu ve službě Apify dle dokumentace [22]	17
5.1	Příklad odeslání e-mailu	62
5.2	Příklad využití služby TransformFailureService	64
5.3	Definice uzlu pro typ <i>array</i>	66
5.4	Kombinace všech uzlů dohromady	67
5.5	Ukázka reprezentace objektu transformačního schématu	67
6.1	GET http://localhost:4002/raw?url=http://10.0.0.6:3005	77
6.2	GET http://localhost:4002/html?url=http://10.0.0.6:3005	77
6.3	Mock server log znázorňuje, že na server přišly pouze dva requesty ve sledovaném období	78
6.4	URL Fetcher log zobrazuje, že většina požadavků byla načtena z cache ve sledovaném období. Nový request byl odeslán na server až po uplynutí 10 sekund	79
6.5	URL Fetcher log pro <i>raw</i> requesty, které nepoužívají Cache-Control (zde ukázáno na příkladu pro ETag)	80
6.6	URL Fetcher log pro <i>html</i> requesty	80
6.7	Chyba při transformaci nevalidní URL	81
6.8	Výsledek unit testů	85

Úvod

Web v současné době tvoří největší kolekci dat a informací na světě. Většina z nich jsou nestrukturovaná, nebo polo-strukturovaná data. Tyto informace jsou pro uživatele snadno čitelné, ale pro stroje obtížněji zpracovatelné.

Motivace pro vývoj aplikace pramení z potřeby automatizovaného sběru dat z internetu. Velká část webů neposkytuje žádné API (anebo jen částečné) a svá data prezentuje ve formě HTML dokumentů. V takovém případě je pro extrakci dat potřeba využít speciálních programů, tzv. *scrapersů*. Tyto nástroje současně mohou umožňovat transformaci dat do vhodného, strojově čitelného formátu (např. JSON, XML aj.).

Cíl práce

V této diplomové práci se zabýváme návrhem a realizací prototypu webové aplikace **MakeAPI**, která je zaměřena na scraping a transformaci dat z jednoduchých HTML stránek. Hlavním cílem práce je vytvoření open-source řešení, které umožní uživatelům snadno získávat, transformovat a integrovat data z různých webových zdrojů. Aplikace poskytuje webové rozhraní pro definici transformačního schématu. K extrahovaným datům následně umožňuje přistoupit skrze API endpoint.

Struktura práce

V kapitole 1 se zabýváme analýzou existujících webových technologií. Zaměříme se zejména na HTTP a jeho techniky používané pro cachování, nastavení CORS aj. Podíváme se také na nástroje pro automatické procházení webu a extrakce dat (crawling a scraping) a metody používané pro omezení těchto tzv. robotů (rate-limiting a Robots Exclusion Protocol).

V následující kapitole 2 se zaměříme na průzkum existujících nástrojů pro extrakci a transformaci dat z webu. Zjistíme, že existuje velké množství způ-

sobů, jakým se dá přistupovat k řešení zadaného problému. Srovnáme vybrané parametry jednotlivých nástrojů (jako je cena, způsob extrakce dat a její konfigurace, podporované formáty aj.) a způsoby, jakým jsou nástroje zpracovány.

Softwarovou analýzu provedeme v kapitole 3. Nejprve provedeme analýzu funkčních a nefunkčních požadavků. Následně provedeme modelování případů užití. Z závěru kapitoly pak provedeme modelování nejdůležitějších procesů aplikace.

V kapitole 4 provedeme návrh aplikace. Nejprve se zaměříme na návrh architektury aplikace, popíšeme jednotlivé komponenty a služby. Zaměříme se detailně na samotný proces transformace dat a reprezentace transformačních schémat. Dále popíšeme diagram nasazení a DB relační model. Na závěr kapitoly popíšeme vybranou metodu pro rate-limiting.

Implementací prototypu se budeme zabývat v kapitole 5. Projekt rozdělíme na dvě části – hlavní aplikaci MakeAPI (poskytuje webové uživatelské rozhraní a API endpoint) a samostatnou službu URL Fetcher (poskytuje stahování dat z webu). Podrobněji popíšeme strukturu kódu a jednotlivé služby. Popíšeme uživatelské rozhraní pro definici transformačního schématu.

Testováním se budeme věnovat v kapitole 6. Testování rozdělíme na manuální a automatizované testy. V rámci manuálního testování ověříme základní funkčnost aplikace (přihlášení, registrace, definice transformačního schématu, extrakce dat, ...). V rámci automatizovaného testování provedeme end-to-end testy, které ověří scénáře průchodu aplikace na základě vybraných případů užití. Současně provedeme také unit testy, které ověřují správnost transformační funkce.

V závěru práce, v kapitole 7, provedeme diskusi a navrhne možná vylepšení aplikace. Zaměříme se na možnosti rozšíření aplikace o další funkce (např. podpora pro další typy transformací, využití AI, aj.).

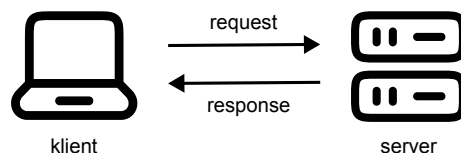
Teorie webu

V této kapitole jsou popsány základní pojmy a technologie, které jsou využívány v této práci. Nejprve je popsán web a základní technologie pro komunikaci a prezentaci obsahu na webu. Následně jsou popsány vlastnosti HTTP protokolu a struktura předávaných zpráv mezi klientem a serverem. Dále jsou popsány různé druhy proxy serverů a jejich využití. Následně je popsána HTTP cache a související HTTP hlavičky. Nakonec je popsán princip rate-limitingu, který slouží k omezení počtu requestů na server.

1.1 Web a základní technologie

Web tvoří rozsáhlá, široce distribuovaná síť vzájemně propojených digitálních informací, služeb a zdrojů. Obsahuje velké množství heterogenních, semi-strukturovaných, redundantních a dynamických dat. [1]

Web slouží jako globální platforma pro sdílení dat mezi účastníky, kteří se dělí do dvou základních rolí - **klienti** a **servery**. Komunikují mezi sebou prostřednictvím výměny zpráv synchronním způsobem. Komunikaci vždy zahajuje klient posláním požadavku a následně čeká, než server vrátí odpověď. [2, str. 3] Komunikace je znázorněna na obrázku 1.1



Obrázek 1.1: Model komunikace klient/server

Zprostředkování komunikace na webu zajišťuje aplikační protokol **HTTP** (Hypertext Transfer Protocol). Zpráva odeslaná z klienta na server se nazývá **request**. Odpověď, kterou posílá server zpátky na klienta, se nazývá

response. Klient (také nazývaný jako *user-agent*) je typicky webový prohlížeč, skrze který uživatel přistupuje k webovým stránkám, avšak klientem může být jakýkoliv program, který odesílá requesty na server. Jde například o roboty, crawlery, webové služby nebo API klienty. [3]

Odpovědí na request je zpravidla **HTML dokument** (avšak odpovědí může být jakýkoliv jiný typ dat, např. obrázek, video, XML dokument), který je zobrazen uživateli v prohlížeči. HTML dokument je textový soubor, který obsahuje semi-strukturovaná data ve formě značek, *HTML tagů*. Tyto tagy tvoří stromovou strukturu, která je základem pro zobrazení dokumentu v prohlížeči. HTML dokument může obsahovat odkazy, přes které se lze navigovat na další dokumenty. [4]

Struktura HTML dokumentu je reprezentována skrze **DOM** (Document Object Model). Jedná se o API pro přístup a manipulaci dokumentu, který je reprezentován stromem s uzly. Každý uzel reprezentuje část dokumentu a může mít další potomky, anebo je přímo listem. [5]

Dokumenty na webu jsou dostupné prostřednictvím **URL** (Uniform Resource Locator). URL jednoznačně identifikuje zdroj v rámci webu. Skládá se z částí: *schéma*, *autorita*, *cesta*, *query* a *fragment*. Schéma určuje protokol, kterým se má přistupovat ke zdroji. Autorita určuje server, na kterém je zdroj dostupný, skládá se z podčástí *userinfo* (uživatelské jméno a příp. heslo), *host* (IP adresa serveru nebo doména) a *portu*. Cesta určuje umístění zdroje na serveru. Query obsahuje parametry, které se předávají serveru. Fragment určuje část zdroje, která má být zobrazena. Struktura URL je zobrazena na výpisu kódu 1.1. [6, kap. 3]

```
foo://example.com:3000/over/there?name=ferret#nose
 \_/  \_____/\_____/\_____/\_/\_/\
 |      |           |           |           |
schéma  autorita   cesta      query    fragment
      |
      \= [ userinfo "@" ] host [ ":" port ]
```

Výpis kódu 1.1: Ukázka URL

1.2 Vlastnosti HTTP

Spojení mezi klientem a serverem je na transportní vrstvě realizováno prostřednictvím TCP¹ (Transmission Control Protocol). Klient se při odeslání requestu připojí na server a po dokončení komunikace, jakmile dostane response, se odpojí. Nové připojení se vytváří pro každý request, ačkoliv existují techniky, které umožňují využít jedno připojení pro více requestů. Ve verzi HTTP 1.1 je možné využít *keep-alive* spojení, které umožňuje nechat spojení

¹Verze HTTP/3 umožňuje také UDP

otevřené pro následné requesty. Ve verzi HTTP 2.0 se využívá multiplexing, který umožňuje využít jedno TCP spojení pro více requestů a odpovědí.

Na transportní vrstvě prochází zprávy skrze *síťová zařízení* (jako jsou routery, switche atd.), která zajišťují navázání a udržení spojení mezi klientem a serverem. Na aplikační vrstvě mohou zprávy po cestě procházet také mezi dalšími *servery*, které se obecně nazývají **proxy**. Proxy může mít různé účely, například load balancing, cachování, rate-limiting atd. (Více v kapitole 1.3 – Proxy.)

Struktura HTTP zpráv je velmi jednoduchá, skládá se ze tří částí: **start-line**, **hlaviček** a **těla**. *Start-line* obsahuje informace o requestu/response, jako je metoda (GET, PUT, ...), verze protokolu, URL zdroje u requestu a status kód (200, 4xx, 5xx, ...) u response. *Hlavičky* obsahují metadata o zprávě, jako jsou například informace o typu obsahu, délce obsahu, cachování, cookies atd. Mezi hlavičkami a tělem zprávy je prázdný řádek. *Tělo* obsahuje samotná data, která jsou přenášena. Zprávy jsou čitelné pro člověka, což umožňuje snadnou kontrolu a ladění komunikace. Příklad HTTP zprávy je zobrazen na výpisu kódu 1.2. [3]

```
GET / HTTP/1.1                                HTTP/1.1 200 OK
Host: make-api.local                          Date: Sat, 09 Oct 2010 14:28:02 GMT
                                              ETag: "51142bc1-7449-479b075b2891b"
                                              Cache-Control: public, max-age=3600
                                              Content-Length: 29769
                                              Content-Type: text/html
```

<!DOCTYPE html>... (zbytek těla)

Výpis kódu 1.2: Ukázka HTTP requestu (vlevo) a HTTP response (vpravo)

Struktura HTTP zpráv umožňuje **snadné rozšíření protokolu** o nové funkcionality pomocí přidání nových hlaviček. [3] Dříve se například používalo rozšíření HTTP hlaviček, které začínaly prefixem *X-*, tímto se často rozlišovalo mezi standardizovanými a nestandardizovanými hlavičkami. Toto se však již dle RFC 6648 [7] nedoporučuje. Může se stát, že některé nestandardizované hlavičky budou v budoucnu standardizovány a získají nový význam. V takovém případě by mohlo dojít ke kolizi, pokud by se používala stejná hlavička, která by měla jiný význam. Vlastní hlavičky lze využít například při ladění komunikace, kdy je potřeba přenášet dodatečné informace, které nejsou součástí standardních hlaviček.

Samotný protokol je **bezstavový**, což znamená, že každý request je nezávislý na ostatních requestech. Server tedy nemá žádné informace o předchozích requestech, které klient poslal. Toto chování je však možné změnit pomocí **cookies**. Cookies jsou malé textové soubory, které jsou uloženy v prohlížeči a jsou odesílány na server společně s requestem. Server je tak schopen identifikovat klienta a udržovat si o něm informace. [3]

1.2.1 HTTP metody

HTTP definuje několik metod (tzv. *HTTP verbs*), které určují, jakým způsobem má být požadavek zpracován. Metody jsou součástí *start-line* requestu. [8]

Mezi nejčastěji používané patří:

GET	základní metoda, používá se pouze získání dat z požadovaného zdroje bez dalších operací na straně serveru
POST	odeslání dat na server, typicky způsobují změnu stavu – <i>side-effects</i> – na straně serveru
PUT	nahrazení obsahu zdroje
DELETE	smazání zdroje
OPTIONS	získání informací o možnostech komunikace s daným zdrojem (využívá se například pro <i>preflight request</i> pro CORS požadavky, viz kapitola 1.6 – CORS protokol)

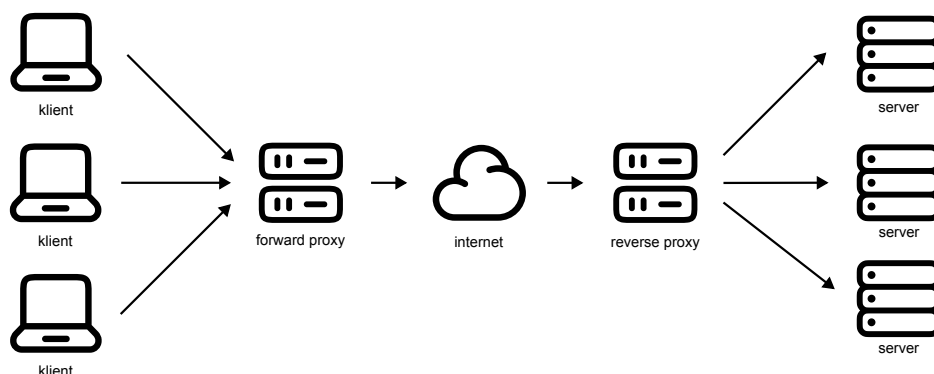
1.2.2 HTTP stavové kódy odpovědi

HTTP response obsahuje číselný kód stavu odpovědi, který informuje o výsledku requestu. Kódy jsou rozděleny do pěti tříd, které se liší svým významem. [9]

1xx – Informational	oznamují, že request byl přijat a zpracovává se
2xx – Successful	oznamují, že request byl úspěšně zpracován
3xx – Redirection	oznamují, že je potřeba přesměrování nebo zvláštní zpracování requestu
4xx – Client Error	oznamují, že request obsahuje chybu
5xx – Server Error	oznamují, že server selhal při zpracování requestu

Mezi základní hlavičky, relevantní pro tuto práci, patří:

200 OK	úspěšná odpověď na GET request
304 Not Modified	zdroj je stále platný (z hlediska HTTP cache)
400 Bad Request	request obsahuje syntaktickou chybu
401 Unauthorized	request vyžaduje autorizaci
403 Forbidden	server odmítá zpracovat request
404 Not Found	zdroj nebyl nalezen
405 Method Not Allowed	metoda není povolena pro daný zdroj
429 Too Many Requests	byl překročen limit počtu requestů (využívané při rate-limitingu)



Obrázek 1.2: Forward-proxy a reverse-proxy

500 Internal Server Error server selhal při zpracování requestu

503 Service Unavailable požadovaný zdroj není momentálně dostupný

1.3 Proxy

Proxy servery zprostředkovávají přístup k obsahu na webu. Proxy může být přímo na počítači uživatele, anebo kdekoli po cestě mezi uživatelským počítačem a koncovým serverem. Proxy servery dělíme na **forward proxy** (častěji pouze *proxy*) a **reverse proxy**. [10]

Forward proxy poskytuje službu **klientovi** (resp. skupině klientů), zejména pak za účelem snížení zátěže sítě. Poskytuje např. DNS, anebo cache. Může skrývat identitu skupiny klientů při odesílání požadavku na server – odeslaný request zvnějšku vypadá, jako by pocházel přímo od proxy. Proxy může doplnit do requestu dodatečné informace prostřednictvím hlaviček, např. o tom, který klient request odeslal.²

Reverse proxy poskytuje službu **serveru** (resp. skupině serverů), zejména pak za účelem zvýšení bezpečnosti, výkonu a škálovatelnosti. Může poskytovat load-balancing, caching, anebo zajišťovat bezpečnostní funkce, jako je např. autentizace, šifrování atd. Často může skupina serverů být schována za jedním reverse proxy serverem, který pak následně rozprostře požadavky na jednotlivé servery uvnitř skupiny, zvnějšku však vypadá, jako by byl pouze jeden server.

Klientem však může být i server v pozici, kdy využívá forward proxy k přístupu na jiné servery. Např. pokud stahuje data z jiného serveru, může využít proxy ke zprostředkování přístupu k datům a jejich cachování.

²Například hlavička **X-Forwarded-For** nebo **Forwarded**, které obsahují IP adresu klienta, nebo **Via**, která obsahuje informace o proxy serveru.

1.4 HTTP Cache

HTTP cache ukládá response asociovaný s daným requestem a znovu použije uloženou odpověď pro následné requesty. Hlavním důvodem pro použití cache je zvýšení rychlosti odezvy a snížení zátěže sítě a serveru. Cache se dělí na dva typy - **privátní** (*private cache*) a **veřejné/sdílené** (*shared cache*). [11]

Private cache je asociována s jedním uživatelem a je dostupná pouze pro něj. Například cache v prohlížeči. Může ukládat soukromé informace určené pouze pro daného uživatele.

Shared cache je dostupná pro více uživatelů. Například cache na proxy serveru. Response cachovaný na takovém serveru může být použitý pro více uživatelů. Rozděluje se na dva typy – **proxy cache** (cache na proxy serveru) a **managed cache** (cache, která je pod kontrolou poskytovatele služby a slouží ke zefektivnění přístupu k obsahu, řadí se sem např. reverse-proxy nebo CDN).

Uložené response mají dva stavy – **fresh** (*čerstvý*) a **stale** (*zastaralý*). Response je ve stavu *fresh*, pokud je stále platný a může být použit pro následné requesty. Response je *stale*, pokud je již neplatný a musí být obnoven. Rozhodčím aspektem je zde **age** (*věk*) response. V rámci response specifikuje server jeho **max-age** (*maximální věk*), který určuje, jak dlouho může být response považován jako *fresh*. Pokud je response starší než *max-age*, je považován za *stale*. [11]

1.4.1 HTTP hlavičky pro kontrolu cache

Hlavičky v requestu a response, které ovlivňují chování cache, jsou následující:

Cache-Control hlavička obsahuje direktivy, které určují, jak má být response cachován. Je součástí requestu i response. Instrukce obsahují například maximální dobu uložení cache (**max-age**), zda-li má být response cachován (**no-store**) aj.

Expires hlavička response obsahuje datum a čas, kdy expiruje *čerstvost* odpovědi. Jedná se o starší hlavičku z HTTP/1.0, která byla nahrazena hlavičkou **Cache-Control**, avšak některé servery ji stále používají.

Vary hlavička response, která určuje seznam hlaviček, které mají být použity k rozlišení response. Využívá se v rámci *content negotiation*, kdy se na základě hlaviček requestu vybírá nejvhodnější reprezentaci obsahu. Například pro hlavičku **Accept-Language** vrací server odpověď na základě požadovaného jazyku dokumentu (např. pro **Accept-Language: cs** může server vrátit jiný obsah než pro **Accept-Language: en**). Hlavička se využívá při cachování pro rozlišení response, klíčem pro identifikaci je tak kombinace URL a hlaviček uvedených v hlavičce **Vary**.

ETag hlavička response obsahuje identifikátor, který jednoznačně identifikuje specifickou verzi odpovědi.

Last-Modified hlavička response obsahuje datum a čas, kdy byl response naposledy změněn. Je méně přesný než *ETag*, používá se jako fallback.

If-None-Match hlavička requestu obsahuje hodnotu získanou z *ETag*, server porovná hodnotu s aktuální verzí a pokud se shoduje, vrátí stavový kód HTTP 304 (Not Modified) a tělo odpovědi vynechá. Tím ulehčuje zátěž sítě.

If-Modified-Since hlavička obsahuje datum a čas, kdy byl response naposledy změněn, funguje obdobně jako *If-None-Match*. Server zkontroluje, zda-li došlo ke změně od zadané doby a pokud ne, vrátí stavový kód HTTP 304 (Not Modified) a tělo odpovědi vynechá.

V případě použití hlaviček *If-None-Match* a *If-Modified-Since* mluvíme o podmíněných požadavcích, tzv. **conditional requests**. Jejich zpracování se od standardních requestů liší tím, že server nemusí vrátit celý obsah odpovědi, ale pouze stavový kód HTTP 304 – Not Modified, pokud server vyhodnotí, že na základě hlaviček je cachovaný obsah na straně klienta stále *fresh*. V takovém případě odešle pouze hlavičky odpovědi a prázdné tělo.³ Tímto ulehčuje zátěž sítě, protože server neodesílá data navíc. [12]

1.4.1.1 Cache-Control

Hlavička Cache-Control určuje, jak mají být odpovědi cachovány jak na straně klienta, tak i na sdílených cache serverech. Obsahuje direktivy, které určují, jak má být response cachován. [13]

Hlavní direktivy, které se v rámci nastavení Cache-Control používají, jsou následující:

max-age určuje, jak dlouho může být response považován za *fresh*. Po uplynutí doby je považován za *stale* a musí být obnoven. Obdobně funguje direktiva **s-maxage**, která je určena pouze pro sdílené cache servery.

no-store zakazuje ukládání response do cache. Response je tedy vždy považován za *stale* a musí být obnoven.

no-cache zakazuje použití response z cache, dokud nebude ověřeno, že je stále platný.

private určuje, že response je určen pouze pro koncového klienta a nesmí být uložen ve sdílené cache.

³Podmíněných hlaviček existuje i více, např. **If-Unmodified-Since** a **If-Match**

public určuje, že response může být cachován i na sdílených cache serverech.

Základní příklady použití hlavičky Cache-Control v rámci response:

- **Cache-Control: max-age=3600** - response může být načtený z cache, pokud je jeho věk menší než 1 hodina (3 600 sekund),
- **Cache-Control: no-cache** - response nesmí být načtený z cache, dokud nebude ověřeno, že je stále platný,
- **Cache-Control: no-store** - response nesmí být uložen do cache,
- **Cache-Control: private, max-age=3600** - určuje, že response může být načtený z privátní cache, pokud je jeho věk menší než 1 hodina. Response je určen pouze pro koncového klienta a nesmí být uložen ve sdílené cache,
- **Cache-Control: public, max-age=3600** - obdobně, ale může být uložen ve sdílené cache.

1.5 Rate-limiting

Rate-limiting je technika, která omezuje počet odeslaných requestů na server. Udává limit, **kolik requestů** může klient odeslat **za určitý časový limit**. (Např. 100 requestů za minutu.) Tato technika se používá pro zabránění zahlcení serveru požadavky od robotů, ať už jako prevence proti DoS, DDoS nebo brute-force útokům, anebo jako omezení crawlerů. [14]

Omezení může být jak na straně samotné aplikace na daném serveru, tak třeba i v rámci reverse-proxy (sdílený limit pro více aplikací/serverů). Rate-limiting může být implementován na různých úrovních identifikace klienta, například na úrovni IP adresy, uživatele, API klíče atd. Po překročení limitu dojde k zablokování přístupu a zpracování dalších requestů je přerušeno, server všechny příchozí requesty zahodí a odpoví stavovým kódem HTTP 429 (Too Many Requests), dokud nedojde k obnovení limitu. [14]

1.5.1 RateLimit hlavička

Ačkoliv není standardizovaná, existuje návrh nové hlavičky **RateLimit**, která by měla sloužit k předávání informací o rate-limitingu. Hlavička by měla obsahovat informace o limitu, počtu zbývajících requestů a časovém limitu. Jedná se náhradu původního návrhu kombinace hlaviček *RateLimit-Limit*, *RateLimit-Remaining* a *RateLimit-Reset*. Můžeme se však setkat i s variantami *X-RateLimit-Limit*, *X-RateLimit-Remaining* a *X-RateLimit-Reset*, které se dle RFC 6648 [7] nedoporučuje používat. Doplnující informace o pravidlech rate-limitingu lze předat v hlavičce *RateLimit-Policy*.

RateLimit hlavička obsahuje následující klíčová slova:

limit (povinná) určuje celkový limit pro daný časový limit, udává se v *jednotkách (quota-units)*, které mohou být vážené, např. pro náročnější požadavek může být vyžadováno více jednotek.

remaining (volitelná) určuje počet zbývajících jednotek, než dojde k omezení přístupu

reset (povinná) určuje časový limit, po kterém se limit obnoví

Příklad použití: `RateLimit: limit=100, remaining=50, reset=60`. (Zde je limit 100 jednotek, zbývá 50 jednotek, než dojde k omezení, a limit se obnoví za 60 sekund.)

1.6 CORS protokol

CORS (Cross-Origin Resource Sharing) protokol umožňuje webovým aplikacím přistupovat ke zdrojům na jiných doménách, než je doména, ze které byla webová stránka stažena. Jedná se o zabezpečení na straně prohlížeče. Serveru umožňuje specifikovat, z jakých zdrojů k němu může být přistupováno. Zdroj – **origin** – je definován jako kombinace *protokolu, domény* (resp. IP adresy) a *portu*. Pokud se zdroj, ze kterého byl odeslán request, liší od zdroje, na který byl request zaslán, jedná se o **cross-origin request** a aplikují se na něj bezpečnostní pravidla.

Důvodem pro zavedení CORS protokolu byla bezpečnost na straně prohlížeče. Pokud by prohlížeč umožňoval přistupovat ke zdrojům na jiných doménách, mohlo by dojít ke zneužití pro útoky typu CSRF (cross-site request forgery), webová stránka by mohla získat citlivá data, ke kterým by jinak neměla přístup. CORS pravidla se aplikují na všechny requesty zasláné pomocí javascriptu, skrze funkce `fetch` nebo `XMLHttpRequest`.

Mechanismus zabezpečení využívá rozšíření HTTP protokolu o nové hlavičky. V jednoduchosti, při cross-origin requestu prohlížeč odešle request s hlavičkou **Origin**, která obsahuje *origin*, ze kterého byl request odeslán. Server může zpět poslat hlavičku **Access-Control-Allow-Origin**, která obsahuje *origin*, kterým je povolen přístup ke zdroji. Prohlížeč vyhodnotí, zda je přístup ke zdroji povolen. Pokud ano, request je dokončen. Pokud ne, request je zablokován. [15]

1.6.1 Hlavičky CORS protokolu

Mezi hlavičky CORS protokolu patří následující:

Origin hlavička requestu obsahující *origin*, ze kterého byl request odeslán.

Access-Control-Allow-Origin hlavička response obsahující *origin*, kterému je povolen přístup ke zdroji. Pokud je povolen přístup všem, obsahuje hodnotu „*“.

Access-Control-Allow-Methods hlavička response obsahující seznam metod, které jsou povoleny pro přístup ke zdroji.

Access-Control-Allow-Headers hlavička response obsahující seznam hlaviček, které jsou povoleny pro přístup ke zdroji.

Access-Control-Expose-Headers hlavička response obsahující seznam hlaviček, které mohou být přístupné z javascriptu.

Access-Control-Allow-Credentials hlavička response, která určuje, zda-li je možné přistupovat ke zdroji pomocí cookies.

Access-Control-Max-Age hlavička response, která určuje, jak dlouho má být cachována informace o tom, zda-li je přístup ke zdroji povolen.

1.6.2 Preflight request

Preflight request je speciální typ CORS requestu, který prohlížeč automaticky odešle před samotným *cross-origin* requestem. Slouží k ověření, zda-li je přístup ke zdroji z hlediska CORS povolen. Od samotného requestu se odlišuje tím, že je odeslán s metodou `OPTIONS` a příslušnými hlavičkami⁴. Server může zpět poslat hlavičky⁵, které určují, zda-li je přístup povolen. [15]

Pokud je request odeslán na stejnou doménu, ze které byla webová stránka stažena, není potřeba provádět preflight request. V opačném případě se takový request odešle automaticky ve většině případů, až na výjimku tzv. *simple requestu*, která splňuje určité podmínky na typ metody (`GET`, `HEAD`, nebo `POST`) a výběru požadovaných hlaviček (povoleny jsou pouze hlavičky `Accept`, `Accept-Language`, `Content-Language` a `Content-Type`⁶). [15]

1.7 Crawling a scraping

Dle [16] je webový **crawler** (nazývaný také jako *spider* nebo *robot*) program, který automaticky prochází webové stránky a stahuje jejich obsah. Rozlišujeme dva typy crawlerů - **univerzální** a **preferenční**. Univerzální crawler prochází webové stránky bez ohledu na jejich obsah, typicky se jedná o *indexery* webových vyhledávačů. Preferenční crawler prochází pouze stránky určitého typu,

⁴`Origin`, `Access-Control-Request-Method` a `Access-Control-Request-Headers`

⁵`Access-Control-Allow-Origin`, `Access-Control-Allow-Methods` a další

⁶Pro *simple request* jsou povoleny pouze `application/x-www-form-urlencoded`, `multipart/form-data` a `text/plain`, tj. pokud je *Content-Type* odpovědi například `application/json`, což je typická situace pro API endpointy, pak se preflight request posílá vždy.

nebo daného tématu. Vedle toho existují také škodlivé crawly, které procházejí webové stránky za účelem získání citlivých dat.

Crawlers tvoří většinu provozu na webu. Aby nedocházelo k přehlcení webových serverů požadavky od robota, je potřeba dodržovat etická pravidla. Pokud by robot procházel stránky příliš rychle, mohl by omezit přístup ostatním uživatelům. Takovým pravidlům se říká **etiketa procházení** (*crawler etiquette*). [16, str. 315]

Každý robot by se měl identifikovat pomocí **User-Agent** hlavičky. Tato hlavička by měla obsahovat informace o robotovi, jako je název, verze a případně i kontaktní informace. [16, str. 315]

```
User-agent: *  
Disallow: /admin
```

Výpis kódu 1.3: Ukázka souboru robots.txt, který zakazuje přístup k adresáři /admin všem robotům.

Etiketa také vyžaduje dodržování pravidel **Robots Exclusion Protocolu**. Jedná se o soubor pravidel, která umožňují webovým správcům ovlivnit chování robotů. Pravidla jsou definována pomocí souboru **robots.txt**, který je umístěn na kořenové cestě domény (př.: <https://www.example.com/robots.txt>). Příklad souboru je zobrazen na ukázce 1.3. Dle RFC 9309 [17] soubor obsahuje seznam direktiv, které určují, které části webu by neměly být procházeny robotem. Robot by měl před procházením stránky zkontrolovat, zda je pro něj přístup povolen. [16, str. 316]

Souvisejícím pojmem je **web scraping**, což je technika, která umožňuje získat strukturovaná data z webových stránek, která by jinak nebyla přístupná (např. skrze API). Jedná se o více cílený proces, kde se zaměřují konkrétní stránky. Data jsou získávána skrze pravidla nebo automatického rozpoznávání a následně převedena do strojově čitelné formy. Doplnuje strukturu do dat, která jinak strukturu nemají (např. z HTML dokumentu). [18]

Rešerše

V této kapitole provedeme rešerši existujících řešení v oblasti scrapingu dat a analýzu jejich vlastností. V závěru pak shrneme poznatky získané z analýzy a vzájemně je porovnáme.

2.1 Analýza nástrojů pro scraping webových dat

V rámci rešerše se zaměříme pouze na takové nástroje, které poskytují uživatelské rozhraní pro zadání parametrů scrapingu a následně umožňují získaná data exportovat. (V analýze se dále nebudeme soustředit na konkrétní knihovny a frameworky, jako je například *Scrapy*⁷ pro Python nebo *Crawlee*⁸ pro JavaScript).

2.1.1 ScrapingBee

ScrapingBee [19] je webová aplikace, která umožňuje uživatelům scrapovat data z webových stránek pomocí volání **univerzálního API endpointu**⁹. Služba poskytuje *Request Builder*, což je jednoduchý nástroj, který umožňuje uživateli specifikovat parametry požadavku pro scrapování. (Ukázka GUI je na obrázku 2.1.) Základním parametrem je URL, kterou chceme scrapovat. Výstupem je pak předgenerovaný kód pro volání API endpointu (buď cURL, anebo pro jazyky Python, JS, PHP a další). Výstup lze však zobrazit i přímo v rámci Request Builderu.

Nástroj umožňuje komplexní nastavení velkého množství parametrů extrakce dat. Výstupem je v základním nastavení HTML kód stránky ze zadaného URL. Lze však specifikovat pravidla – **extract rules** – kterými lze definovat strukturu výstupních dat. Definice se zadává prostřednictvím textového pole ve formátu JSON, ukázka formátu je znázorněna na výpisu kódu

⁷<https://scrapy.org/>

⁸<https://crawlee.dev>

⁹GET <https://app.scrapingbee.com/api/v1/?url=...>

2.1. Pro výběr dat se využívají *selektory*, které určují elementy na stránce, ze kterých se mají data extrahovat. Selektory mohou být buď ve formě CSS selektorů, nebo XPath.

V základním nastavení se pro stažení dat použije jednoduchý GET request, avšak služba umožňuje zpracovat data i z webových stránek, které jsou generovány pomocí JavaScriptu. Pro takový typ scrapingu služba umožňuje další nastavení předzpracování dat z webové stránky v rámci headless browseru. Uživatel může například určit, jak dlouho se má čekat na načtení stránky, nebo dokud na stránce není viditelný zadaný element. Zároveň může specifikovat **instrukce** (*JS scenario*), které se mají provést před zpracováním stránky. Ty se zadávají jako seznam příkazů ve formátu JSON.

Služba je placená, ale umožňuje bezplatnou zkušební verzi. V rámci placené verze poskytuje například i řešení proti blokování přístupu robotů na webové stránky. Služba má velmi dobře zpracovanou dokumentaci, celkově je kvalitně zpracovaná a uživatelsky přívětivá.

Nevýhodou je, že uživateli neumožňuje vygenerované parametry pro API endpoint ukládat nebo nějak organizovat. Výstupy služby jsou určeny pro integraci do jiných projektů jako součást jejich kódu.

```
{
  "title" : "h1",
  "subtitle" : "#subtitle",
  "articles": {
    "selector": ".card",
    "type": "list",
    "output": {
      "title": ".post-title",
      "link": {
        "selector": ".post-title",
        "output": "@href"
      },
      "description": ".post-description"
    }
  }
}
```

Výpis kódu 2.1: Ukázka definice *extract rules* pro extrakci dat z blogu ve ScrapingBee

2.1.2 Apify

Apify [20] je platforma pro tvorbu, nasazení a správu webových scraperů. Dle dokumentace [21] umožňuje uživatelům vytvářet **aktéry** (*actors*), což jsou

microservices, které umožňují provádět širokou škálu automatizovaných operací. Mohou běžet pár sekund, ale třeba též několik hodin, nebo i neomezeně dlouho. Může provádět jednoduché operace (jako vyplnění formuláře na webové stránce) nebo složité operace (jako je například crawling a extrakce dat z celého webu).

Aktéry jsou vytvářeny pomocí *Apify SDK*, což je knihovna pro jazyk JS a Python. V rámci webové aplikace lze zadat instrukce pro vytvoření aktéra, který se následně spustí v rámci *Apify cloudu*. Uživatelé mohou využít také existujících aktérů, které jsou dostupné v *Apify Store*. Tento princip umožňuje velmi komplexní zpracování dat z velké škály předdefinovaných zdrojů.

Pro scraping konkrétních webových stránek lze využít speciální aktér **Web Scraper** [22], který umožňuje v rámci své konfigurace zadat definici funkce `pageFunction`, což je JS kód, který se spustí pro každou načtenou stránku. Umožňuje také provádět crawling a procházet odkazy na zadaných URL a přidávat je do fronty pro další zpracování. Příklad konfigurace aktéra je na výpisu kódu 2.2.

```
async function pageFunction(context) {
  const $ = context.jQuery;
  const pageTitle = $('title').first().text();

  // Print some information to actor log
  context.log.info(`${context.request.url}: ${pageTitle}`);

  // Manually add a new page to the scraping queue.
  await context.enqueueRequest({url: 'http://example.com'});

  // Return an object with the data extracted from the page.
  // It will be stored to the resulting dataset.
  return {
    url: context.request.url,
    pageTitle,
  };
}
```

Výpis kódu 2.2: Ukázka konfigurace Web Scraperu ve službě Apify dle dokumentace [22]

Výsledek zpracování je k dispozici v rámci dashboardu aktéra, kde je možné si ho prohlédnout, nebo exportovat do souboru. Výstup je dostupný taky skrze API. Ukázka GUI je na obrázku 2.2.

Apify je placená služba. Poskytuje širokou škálu funkcí, umožňuje vybrat z velké nabídky předpřipravených aktérů, avšak pro jednoduché scrapování dat je její použití poměrně komplikované. Pro definování výstupní struktury

je potřeba znalost JavaScriptu.

2.1.3 Diffbot

Diffbot [23] je služba, která umožňuje uživatelům získávat data z webových stránek pomocí **automatického strojového učení**. Uživatelé zadávají URL, ze které se mají data získat, a Diffbot automaticky analyzuje strukturu stránky a definuje, jaká data se mají získat. Výstupem je pak JSON soubor, který obsahuje získaná data. Ukázka GUI je na obrázku 2.3.

Výhodou aplikace je, že pro základní použití není potřeba znalost HTML. Uživatelé mohou využít z nabídky předdefinovaných typů obsahu (například seznam, článek, produkt atd.). Pro pokročilejší použití služba umožňuje také vytvořit vlastní typ obsahu pomocí **Custom API**, což je nástroj, který umožňuje uživatelům definovat vlastní typy obsahu, vč. specifikace dalších parametrů requestů (využití proxy, nastavení cookies, atp.)

Služba je placená, ale umožňuje 2-týdenní přístup zdarma. Poskytuje velmi jednoduché UI. Výsledek transformace lze exportovat do souboru, nebo získat přímo skrze univerzální API¹⁰.

2.1.4 Import.io

Import.io [24] je webová aplikace, která umožňuje uživatelům získávat data z webových stránek v podobě tabulek a datových sloupců. Uživatelé zadávají URL, ze které se mají data získat, a Import.io automaticky analyzuje strukturu stránky a vytvoří definici, jaká data se mají získat. Výstupem je pak tabulka, která obsahuje získaná data. Ukázka GUI je na obrázku 2.5.

Aplikace umožňuje editaci struktury skrze grafické rozhraní. Uživatelé mohou přidávat nové sloupce, nebo upravovat existující, pomocí vytvoření *patternu* skrze výběr elementů na dané stránce, která je vykreslena v rámci UI. Použití je vhodné pro tabulková data.

Službu nebylo možné vyzkoušet, protože nenabízí zkušební verzi. Pro analýzu jsme vycházeli z ukázek tutoriálu [25].

2.1.5 Octoparse

Octoparse [26] je desktopová aplikace, která poskytuje komplexní řešení pro scrapování dat z webových stránek. Pro extrakci umožňuje výběr z **předdefinovaných šablon** pro často navštěvované weby různých kategorií (např. pro e-shopy, jako je Amazon, eBay, Etsy, pro ubytovací služby, jako je Booking, AirBnB, TripAdvisor, sociální platformy, jako je Facebook, Instagram, Twitter atd.). Aplikace poskytuje také **Advanced Mode**, v rámci kterého lze specifikovat pravidla pro extrakci dat z libovolného webu.

¹⁰<https://api.diffbot.com/v3/analyze?url=...&token=...>

Pravidla extrakce se definují v rámci úkolů (*tasks*). Uživatel specifikuje seznam URL adres a následně definuje **akce** (*actions*), které se mají provést pro každou stránku. Akce mohou být například kliknutí na element, vyplnění formuláře, nebo extrakce dat. Aplikace využívá vykreslení webové stránky, v rámci které lze jednoduše naklikat požadované operace. *Akce* prezentuje formou grafu, kde uzly představují jednotlivé akce a hrany představují jejich návaznost. Ukázka GUI je na obrázku 2.6.

Vzhledem k tomu, že aplikace je dostupná pouze pro systémy Windows a MacOS, nebylo možné ji vyzkoušet. Pro analýzu jsme vycházeli z ukázek tutoriálu [27].

Jedná se o placenou službu, která umožňuje komplexní scraping dat přes více URL. V rámci aplikace je možné využít *local extraction*, která provede extrakci dat lokálně na straně uživatele. Využít lze také *cloud extraction*, kdy se úkol zpracuje na straně serveru. Umí také nastavit časovač, jak často se má extrakce provádět. Umožňuje přístup skrze API.

2.1.6 ParseHub

ParseHub [28] je desktopová aplikace pro Windows, MacOS a Linux. Umožňuje komplexní procházení a získávání dat z webových stránek z výchozí zadané URL. Umožňuje navigaci skrze odkazy na stránce a jejich následné zpracování. Chování extrakce se zadává skrze **akce** (*actions*), což je kombinace instrukcí (např. navigace na odkaz, scrollování na stránce, přesun na další stránku), podmíněných skoků a cyklů a výběru prvků pro extrakci. Umožňuje vytvářet šablony (*templates*), které umožňují seskupovat akce dohromady (např. šablona pro extrakci hodnot z podstránky detailu produktu). Toto umožňuje tvorbu pokročilých transformací. Ukázka GUI je na 2.7.

Zadávání akcí probíhá skrze grafické rozhraní, v rámci kterého je zobrazena cílová stránka. Uživatelé mohou jednoduše naklikat požadované akce přímo na vykreslené webové stránce. Výstupní struktura je určena použitými akcemi a je spjatá s pojmenováním výběrových (*select*) a extrakčních (*extract*) akcí. Příklad převedení akcí na výstupní strukturu je na obrázku 2.8.

Služba je placená, ale poskytuje omezenou bezplatnou verzi. Výstup je možné exportovat do souboru, nebo získat přímo skrze API. Aplikace poskytuje množství tutoriálů, které umožňují uživatelům se s nástrojem rychle seznámit.

2.1.7 Portia

Portia [29] je webová aplikace, která umožňuje crawling a scraping dat z webových stránek. Aplikace organizuje extrakce do projektů, v rámci kterých lze vytvořit **spider**, který určuje, jaké URL se mají procházet. Následně umožňuje specifikovat **sample**, ukázkový příklad webové stránky, kterou využije

jako vzorek pro extrakci z podobných stránek. Lze specifikovat různé typy dat (např. text, číslo, datum, obrázky, odkazy atd.).

Portia neposkytuje hostovanou verzi, ale umožňuje stáhnout zdrojový kód a spustit nástroj lokálně. Aplikace pro scraping a crawling využívá framework *Scrapy*. Pro spuštění extrakce lze využít příkaz `portiacrawl PROJECT_PATH`. Možné je také exportovat vygenerovaný crawler přímo jako Scrapy projekt.

Aplikace je bezplatná a je **open-source**. Uživatelsky je aplikace subjektivně obtížná na použití a definování transformace není příliš intuitivní. Zadávaní pravidel extrakce probíhá pomocí výběru elementů na stránce, zobrazené v rámci UI. Ukázka GUI je na obrázku 2.9.

2.1.8 Další nástroje

Existují i další nástroje, které umožňují scrapování dat z webových stránek. V této sekci uvedeme příklady, kterými jsme v rámci prvotní analýzy dále podrobněji nezabývali.

NoCoding Data Scraper [30] – doplněk pro prohlížeč Chrome. Umožňuje vytvářet transformace s využitím technologie AI, ale zároveň umožňuje i ruční úpravu, avšak definice vlastních pravidel je složitější. Jedná se o placený doplněk a jeho základní použití je omezené, zejména co se týče exportu dat, umožňuje pouze CSV.

Web Scraper [31] – doplněk pro prohlížeč Chrome a Firefox. Jeho uživatelské rozhraní je však poměrně neintuitivní a doplněk působí nedokončeně, proto jsme se jím dále nezabývali.

ScrapeStorm [32] – desktopová aplikace, podobná Octoparse. Umožňuje uživatelům vytvářet *tasky*, které určují interakce s webem a následně definují pravidla extrakce dat. Aplikace je placená, ale umožňuje bezplatnou zkušební verzi. Extrakce probíhá na straně klienta v rámci aplikace

ScrapingAnt [33] – webová aplikace, která je podobná ScrapingBee. Umožňuje uživatelům scrapovat data z webových stránek pomocí volání univerzálního API endpointu. Strukturu dat je schopný detekovat pomocí AI. Služba je placená, ale umožňuje bezplatnou zkušební verzi.

ScraperApi [34] – webová aplikace, která umožňuje scraping dat z více URL, avšak slouží spíše pro crawling a sbírání dat čistě v HTML podobě, další zpracování a extrakci dat je potřeba provést samostatně.

ScrapeNinja [35] – webová aplikace, která je podobná ScrapingBee. Umožňuje využít centrální API endpoint pro scrapování dat. Strukturu výstupních dat lze specifikovat pomocí JavaScriptového snippetu, který je součástí requestu.

2.2 Shrnutí analyzovaných nástrojů

Pro scraping dat z webových stránek existuje velké množství nástrojů. V rámci analýzy jsme se zaměřili na takové nástroje, které poskytují GUI pro zadávání parametrů scrapingu. V tabulce 2.1 jsou shrnuty základní parametry analyzovaných nástrojů.

Soustředili jsme se zejména na způsob zadávání pravidel pro extrakci dat. Všechny zkoumané nástroje umožňují specifikovat, jaká data se mají získat, avšak způsob zadávání pravidel se liší. Většina nástrojů umožňuje zadávat pravidla pomocí výběru elementů na stránce, která je vykreslená v rámci GUI. Některé nástroje umožňují zadávat pravidla pomocí JSON, nebo JavaScriptu.

Nástroje, které poskytují **GUI pro výběr elementů** na stránce, jsou méně flexibilní a více omezené v porovnání s ostatními nástroji (které umožňují zadání skrze JSON nebo kód). Jejich výhodou je však jednoduchost použití a možnost řešit některé časté problémy, jako je například procházení webu skrze stránkování nebo interakce s webovou stránkou (kliknutí na tlačítko, otevření odkazu, atp.). Jejich nevýhodou jsou však problémy s vykreslením stránky v rámci GUI, zejména pak v případě webových aplikací, kdy je vykreslování ovlivněné CORS protokolem a zobrazená stránka tak často vykazuje chyby při vykreslení (CORS často blokuje načtení obrázků, CSS stylů apod.).

Nejvíce flexibilní jsou pak nástroje, které umožňují zadávat pravidla skrze JavaScript (nebo jiný jazyk), který dovoluje zpracovávat i velmi komplexní scénáře a krajní případy.

Vlastní kategorií jsou pak nástroje, které využívají **AI** pro automatickou detekci struktury dat. Tyto nástroje jsou velmi jednoduché na použití, avšak jejich použití je omezené na jednoduché případy, kdy je možné využít předpřipravené šablony. Tyto nástroje jsou méně transparentní, neboť uživatel má menší možnost ovlivnit, jaká data se mají přesně získat.

Většina nástrojů také poskytuje další pokročilé funkce, vedle samotného scrapingu dat z webu. Umožňují například procházení více URL najednou (crawling), zpracování webů generovaných pomocí JavaScriptu, nebo řešení problémů s omezením přístupu robotů na webové stránky a využití proxy serverů z různých zemí, pro obcházení geograficky omezených webů.

Pro samotnou extrakci dat využívají služby různé přístupy. Dají se rozdělit na dvě kategorie – ty, které zpracovávají scraping skrze **univerzální API endpoint** (využívají jeden vstupní bod pro všechny extrakce) a ty, které umožňují **seskupovat a organizovat extrakce** do projektů, aktérů, nebo jiných entit (lze organizovat data v rámci aplikace).

Dále lze aplikace rozdělit podle toho, zda-li umožňují zpracovávat pouze jednu, anebo více URL najednou. S tím také souvisí, jestli je zpracování **synchronní** anebo **asynchronní**. U synchronního zpracování dostaneme data

rovnou – např. při volání API endpointu dojde ke stažení cílové stránky, jejího načtení a extrahování dat v rámci jednoho volání. U asynchronního zpracování je potřeba nejprve vytvořit *úkol* (task, aktér, spider, apod.), který definuje úlohu pro zpracování extrakce a následně nechat službu tuto úlohu zpracovat. Výsledek je dostupný až po dokončení zpracování celé úlohy.

Téměř všechny nástroje jsou **placené**, což je dáno tím, že provozování služby je nákladné, zejména pak v případě zpracování JavaScriptových webů, které vyžadují načtení v prohlížeči. Většina nástrojů však poskytuje bezplatnou zkušební verzi. Některé nástroje poskytují také omezenou verzi zdarma, která je však omezená počtem extrakcí, nebo počtem stránek, které lze zpracovat. Pouze jeden nástroj (Portia) je dostupný zdarma a je **open-source**.

Všechny nástroje umožňují exportovat výsledky do souboru, nebo získat přímo skrze API. Nejčastějšími formáty jsou **JSON**, některé umožňují export do XML, CSV, Excelu nebo HTML.

Co se týče technického zpracování, jednotlivé nástroje jsou převážně na dobré úrovni zpracování. Subjektivně nejlépe vychází nástroj *ScrapingBee*, který poskytuje velmi intuitivní GUI pro zadání transformace. Jeho nevýhodou je však nemožnost organizovat extrakce do projektů nebo skupin. Pro specifikaci extrakce využívá JSON, kde je nutné znát jeho strukturu, aby bylo možné pravidla dobře definovat. Nicméně nástroj poskytuje detailní dokumentaci s příklady a použití je tak poměrně snadné. Je určený zejména pro integraci do jiných aplikací, poskytuje předgenerované kusy kódu pro různé programovací jazyky. Nevýhodou je, že se jedná o placenou službu a není open-source.

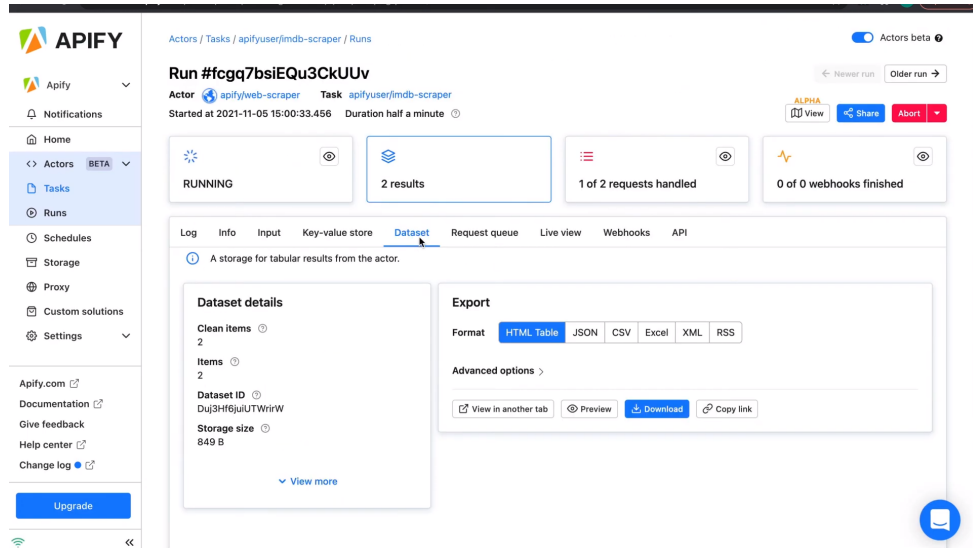
Subjektivně horší zpracování má nástroj *Portia*, který je sice open-source, ale je obtížnější na použití a není tak intuitivní. Jedná se však o jediné open-source řešení napříč zkoumanými nástroji, tudíž by mělo smysl vytvořit nový nástroj, který by byl také open-source, ale zároveň by měl lepší zpracování a byl by jednodušší na použití.

2.2. Shrnutí analyzovaných nástrojů

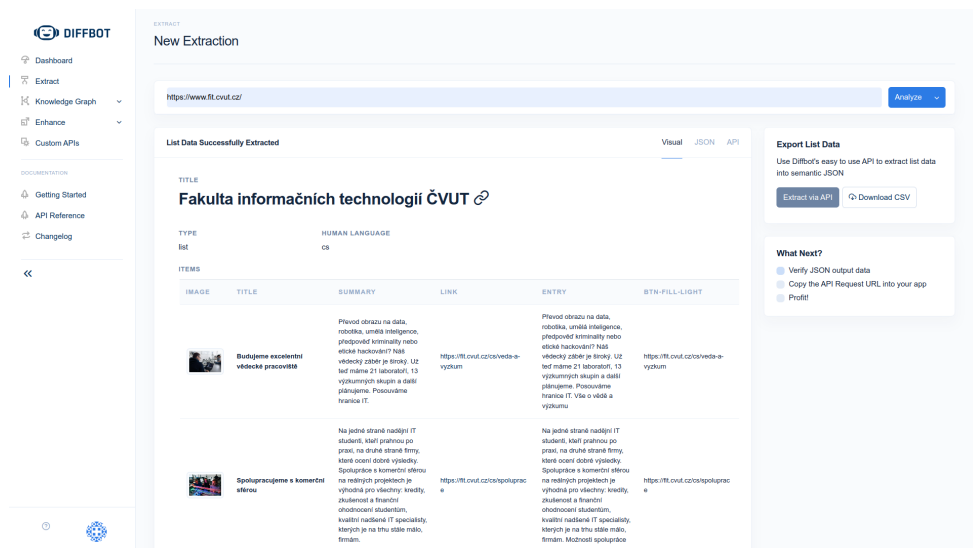
	ScrapingBee	Apify	Diffbot	Import.io	Octoparse	ParseHub	Portia
Typ	Webová aplikace	Webová platforma	Webová aplikace	Webová aplikace	Desktopová aplikace (Win, MacOS)	Desktopová aplikace (Win, Linux, MacOS)	Webová aplikace (nutno nasadit lokálně)
Cena	Placená (poskytuje 1000 kreditů na vyzkoušení zdarma)	Placená (poskytuje omezenou verzi zdarma s kreditem 5 \$ každý měsíc)	Placená (poskytuje 2-týdenní zkušební verzi)	Pouze placená verze	Placená (poskytuje omezenou verzi zdarma)	Placená (poskytuje omezenou verzi zdarma)	Zdarma
Open-source	Ne	Ne	Ne	Ne	Ne	Ne	Ano
Zpřístupnění dat skrze API	Ano	Ano	Ano	Ano	Ano	Ano	Ne
Způsob extrakce	Univerzální API endpoint	Vytváření aktérů	Univerzální API endpoint	Vytváření extraktorů	Vytváření tasků	Vytváření projektů	Vytváření spiderů
Procházení více URL najednou	Ne	Ano	Ne	Ano	Ano	Ano	Ano
Zpracování výsledků transformace	synchronně	asynchronně	synchronně	asynchronně	asynchronně	asynchronně	asynchronně
Výstupní formát dat	JSON, HTML	JSON, XML, CSV, Excel, RSS	JSON, CSV (CSV je pouze pro seznamy/tabulková data)	JSON, CSV	JSON, CSV, Excel, HTML	JSON, CSV	JSON, XML, CSV (Odpovídá formátům exportu ze Scrapy)
Způsob specifikace struktury dat	JSON	JS kód	Automaticky podle AI	Tvorba datových sloupců na náhledu webové stránky	Interaktivní na náhledu webové stránky	Interaktivní na náhledu webové stránky	Interaktivní na náhledu webové stránky
Podpora interakcí s webovou stránkou	Ano, seznam instrukcí, které se mají provést před extrakcí	Ano, v rámci JS kódu	Ne	Ne	Ano	Ano	Ne
Lokální spouštění extrakce	Ne	Ne	Ne	Ne	Ano, v rámci aplikace	Ne	Ano, skrze příkazový řádek
Podpora zpracování JS webů	Ano	Ano	Ano	Ano	Ano	Ano	Ano
Subjektivní zpracování	Velmi kvalitní, moderní zpracování a intuitivní prostředí, dobře zpracovaná dokumentace	Velmi univerzální nástroj, vhodný pro široké spektrum úkolů	Jednoduché UI, snaží se maximálně zjednodušit extrakci, avšak pokročilejší parametrisace je obtížnější	Méně uživatelsky přívětivé, mírně nepřehledné	Složitější UI, které podporuje řadu komplexnějších interakcí	Jednoduché UI, relativně snadné na použití	Působí nepřehledně, vytvoření pravidel extrakce je náročné

Tabulka 2.1: Shrnutí analýzy nástrojů pro scrapování dat

2.2. Shrnutí analyzovaných nástrojů

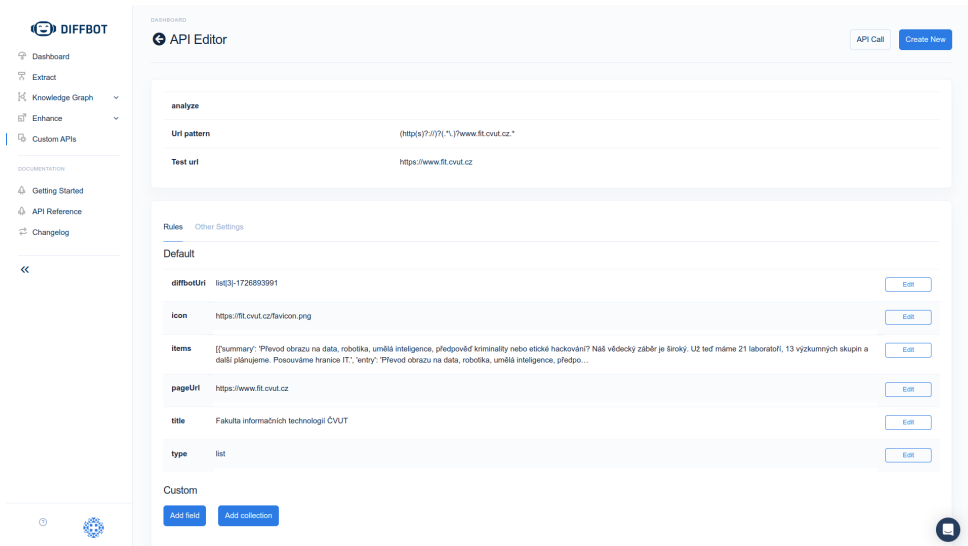


Obrázek 2.2: Dashboard aktéru ve službě Apify

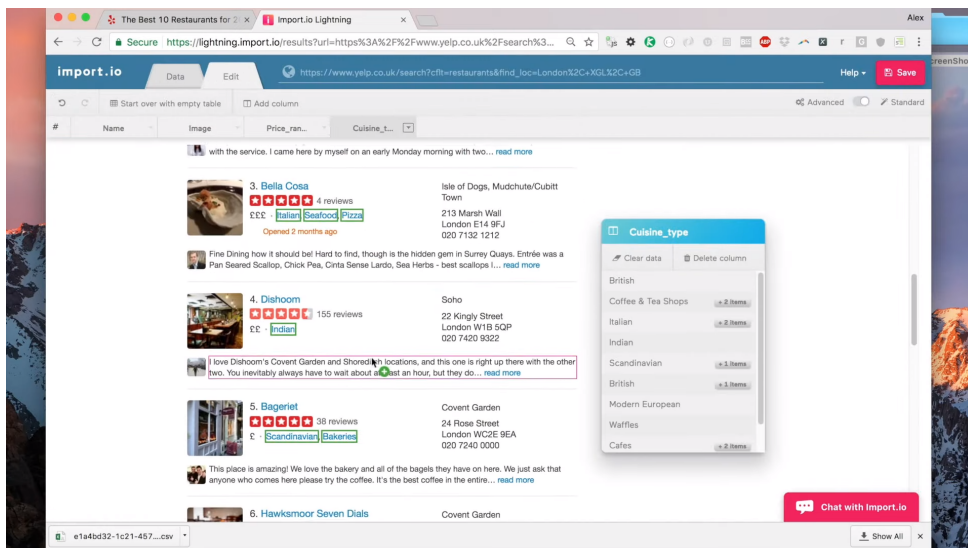


Obrázek 2.3: Ukázka extrakce v aplikaci Diffbot

2. REŠERŠE

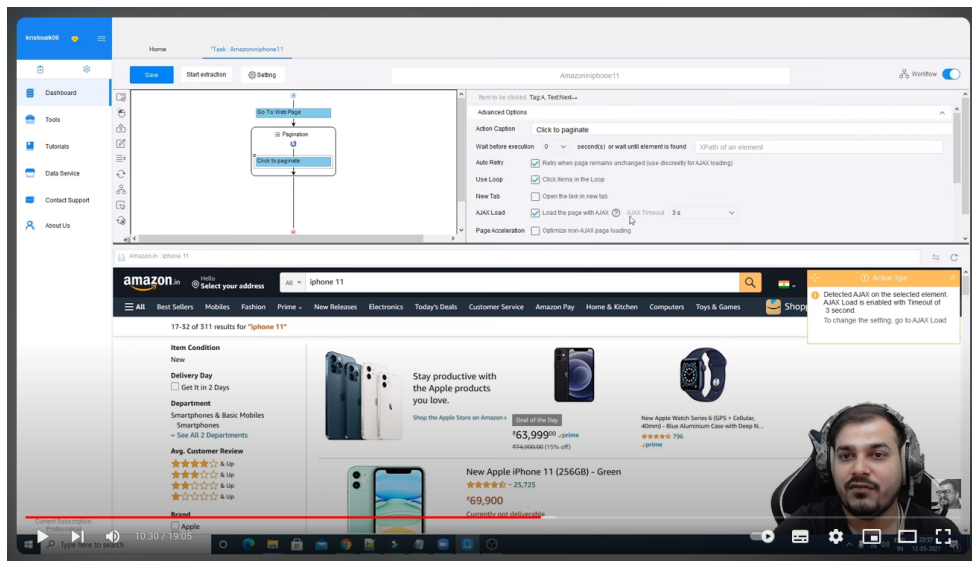


Obrázek 2.4: Ukázka specifikace vlastních pravidel v rámci *Custom API* ve službě Diffbot



Obrázek 2.5: Ukázka extrakce datových sloupců v aplikaci *Import.io*

2.2. Shrnutí analyzovaných nástrojů

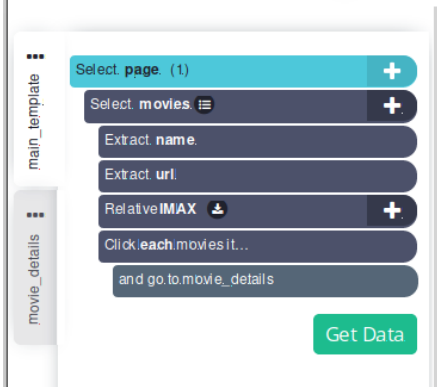


Obrázek 2.6: Ukázka vytváření akcí ve službě *Octoparse*

rating	character_name	character_actor	missing data
9.2	Andy Dufresne	Tim Robbins	
9.2	Ellis Boyd "Red" Redding	Morgan Freeman	
9.2	Warden Norton	Bob Gunton	
9.2	Heywood	William Sadler	
9.2	Captain Hadley	Clancy Brown	
9.2	Tommy	Gil Bellows	
9.2	Bogs Diamond	Mark Rolston	
9.2	Brooks Hatlen	James Whitmore	
9.2	1946 D.A.	Jeffrey DeMunn	

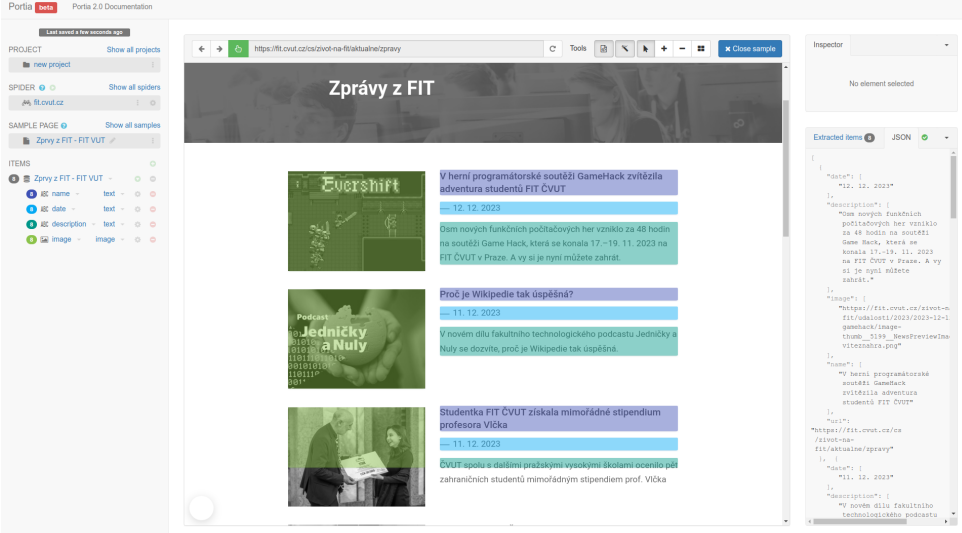
Obrázek 2.7: Ukázka tutoriálu ve službě *ParseHub*

2. REŠERŠE



```
{ "movies": [  
  {  
    "name": "The Shawshank Redemption",  
    "url": "https://parsehub.com/sandbox/  
      moviedetails?movie=  
        The%20Shawshank%20Redemption",  
    "IMAX": "06:00 PM",  
    "rating": "9.2",  
    "character": [  
      { "name": "Andy Dufresne",  
        "actor": "Tim Robbins" },  
      { "name": "Ellis Boyd Redding",  
        "actor": "Morgan Freeman" }  
    ]  
  },  
],  
}]
```

Obrázek 2.8: Ukázka struktury extrahovaných dat *ParseHub*



```
{  
  "date": "  
    "12. 12. 2023"  
  },  
  "description": "  
    "V novém dílu fakultního technologického podcastu Jedničky a Nuly se dozvíte, proč je Wikipedia tak úspěšná."  
  },  
  "image": "  
    "https://fit.cvut.cz/fit-na-fakultne/zpravy/  
      gamehack/image/  
      thumb_015_005_PreviewIma-  
      viteznicna.png"  
  },  
  "name": "  
    "V novém dílu fakultního  
      podcastu  
      soutěží GameHack  
      vítězila adventura  
      studentů FIT CVUT"  
  },  
  "url": "  
    "https://fit.cvut.cz/cs/  
      /11907-na-  
      fit/fakultne/zpravy"  
  },  
  "date": "  
    "11. 12. 2023"  
  },  
  "description": "  
    "V novém dílu fakultního  
      technologického podcastu"  
  }  
}
```

Obrázek 2.9: Ukázka výběru dat ve službě *Portia*

Analýza

V této kapitole je provedena analýza požadavků na aplikaci. Nejprve je popsán rozsah projektu a jeho cíle. Poté jsou uvedeny funkční a nefunkční požadavky na aplikaci a modelování případů užití. V poslední části kapitoly jsou analyzovány hlavní procesy, které budou probíhat v rámci aplikace.

3.1 Popis projektu

Projekt **MakeAPI** je webová aplikace, která umožňuje uživatelům extrahovat data z webových stránek ve strukturované podobě. Rozsah projektu zahrnuje front-endovou část s UI pro zadání definice transformace a back-endovou část, která zajišťuje zpracování a přístup k transformovaným datům skrze API.

3.1.1 Slovník pojmů

V této části popíšeme základní pojmy a termíny, které jsou používány v rámci projektu.

Projekt – organizační celek, který obsahuje kolekci *dokumentů* vytvořených uživatelem. Na úrovni projektu lze vytvořit *access tokeny*, které umožňují přístup k datům skrze *API endpoint*.

Dokument – obsahuje informaci o URL pro extrakci dat a další parametry, jako je typ strategie stahování (buď jako jednoduchý *fetch*, anebo s využitím *headless* browseru), nastavení *striktní transformace*, či ovlivnění parametrů cachování. Každý dokument má přiřazené *transformační schéma*.

Transformační schéma – specifikace transformace, která určuje, jaké prvky se mají extrahovat ze zdrojového dokumentu staženého z URL, určené v nastavení dokumentu. Jedná se o stromovou strukturu, kde jednotlivé

3. ANALÝZA

uzly specifikují extrakci konkrétních prvků pomocí výběru CSS selektorem. Transformační schéma lze specifikovat v aplikaci pomocí UI.

Transformace – proces, kterým se získají data ze zdrojového dokumentu na základě *transformačního schématu*. Výstupem transformace je JSON dokument, který obsahuje extrahovaná data. K výsledku transformace lze přistupovat skrze *API endpoint*.

API endpoint – rozhraní, které zpřístupňuje transformovaná data skrze API.

Selhání transformace – informace o tom, zda-li došlo k selhání transformace. Selhání může být způsobeno nevalidním *transformačním schématem*, chybnými zdrojovými daty, anebo chybou při stahování zdrojových dat (např. špatné URL, anebo nedostupnost zdrojového dokumentu). Pokud dojde k opakovanému selhání transformace při přístupu skrze API, dojde k zablokování další transformace, dokud uživatel neopraví chybu.

Striktní transformace – nastavení, zda-li se mají interpretovat chyby při transformaci jako *selhání transformace*. (Např. pokud nějaký uzel specifikovaný zadaným selektorem vrátí více prvků, než kolik očekává.)

Access token – textový řetězec, který slouží k autentizaci uživatele při přístupu k datům skrze *API endpoint*. Každý token má nastavenou expiraci, po které token přestane být platný. (Může být i neomezeně dlouho.) Zároveň je možné specifikovat povolené *origins* (ve smyslu CORS) pro požadavky odeslané na *API endpoint*.

3.2 Analýza požadavků

V této kapitole jsou uvedeny funkční a nefunkční požadavky na aplikaci, rozdělené do několika skupin podle funkcí, které má aplikace splňovat.

3.2.1 Funkční požadavky

1. Autentizace a autorizace

FR1.1: Registrace a přihlašování uživatelů

Aplikace umožní vytvořit účet a přihlásit se pomocí emailu a hesla.

2. Správa projektů

FR2.1: Správa projektů uživatele

Aplikace umožní organizaci do *projektů*. Projekt obsahuje kolekci *dokumentů* pro transformaci.

Uživatel bude mít možnost vytvořit nový projekt, upravit jej, anebo smazat. V případě smazání projektu se smažou i všechny jeho dokumenty. V rámci projektu umožní přidání, editaci a mazání *access tokenů* pro přístup k datům skrze API.

FR2.2: Správa dokumentů v projektu

Uživatel bude mít možnost přidat nový dokument do projektu, upravit jej, anebo smazat. Dokument obsahuje URL pro extrakci a parametry pro scraping dat.

3. Scraping dat

FR3.1: Extrakce dat ze zadané URL

Aplikace umožní specifikovat zdrojovou URL pro extrakci dat.

FR3.2: UI pro definici transformačního schématu

Aplikace umožní specifikovat transformační schéma pomocí UI, v rámci kterého bude možné určit, jaké prvky se mají extrahovat ze zdrojového dokumentu na zadané URL. Aplikace bude kontrolovat validitu transformačního schématu.

FR3.3: Náhled extrahovaných dat

Aplikace provede transformaci dat na základě zadaného transformačního schématu a zdrojových dat na zadané URL a zobrazí výstup ve formátu JSON.

FR3.4: Nastavení způsobu stahování dat

Aplikace umožní specifikovat způsob stahování dat. Buď pomocí *jednoduchého HTTP requestu* (vhodné např. pro statické weby), anebo pomocí *headless browseru* (vhodné pro složitější weby, které se vykreslují pomocí JS).

FR3.5: Nastavení parametrů cachování

Aplikace umožní nastavit parametry hlavičky *Cache-Control* pro dotazy poslané skrze API endpoint. Uživatel bude mít možnost specifikovat direktivu *max-age*.

FR3.6: Varování v případě omezení podle robots.txt

Aplikace zobrazí uživateli varování, pokud pravidla načtená ze souboru robots.txt pro doménu ze zadané URL zakazují přístup k datům. Aplikace také zobrazí upozornění, pokud se nepodaří načíst informace z robots.txt.

4. API endpoint

FR4.1: Správa *access tokenů* pro přístup k datům

Aplikace umožní přidání, editaci a mazání *access tokenů*. Při vytvoření se vygeneruje náhodný řetězec, který si uživatel musí uložit.

3. ANALÝZA

(Aplikace nebude ukládat access token v plaintextu, ale pouze jeho hash, proto není možné si ho zobrazit později.)

Uživatel bude mít možnost editovat popisek a expiraci tokenu. Zároveň v rámci access tokenu bude možné specifikovat povolené *origins* (ve smyslu CORS) pro požadavky odeslané na API endpoint.

FR4.2: **Zpřístupnění extrahovaných dat skrze API endpoint**

Aplikace zpřístupní extrahovaná data skrze API endpoint. Přístup k datům bude chráněn pomocí access tokenu. HTTP odpovědi budou obsahovat HTTP hlavičky pro cache, CORS a rate-limiting.

FR4.3: **Zobrazení chyby v případě selhání extrakce dat**

Pokud se nepodaří extrahovat data, není validní extrakční schéma, nebo se nepodaří stáhnout zdrojová data, aplikace zobrazí chybu.

FR4.4: **Omezení počtu requestů na API endpoint**

Aplikace omezí počet requestů na API endpoint pomocí techniky *rate-limitingu*, omezení bude na dvou úrovních: pro *access token* a pak globálně pro *uživatele*. Výchozí hodnoty bude možné specifikovat v rámci konfigurace aplikace.

5. Reporting

FR5.1: **Zablokování další extrakce v případě selhání**

Pokud se nepodaří extrahovat data, aplikace zablokuje další extrakci dat a zobrazí chybu. Zablokování proběhne v případě opakovaného přístupu k API endpointu, který skončil chybou. Zablokování proběhne až po uplynutí minimálního časového limitu, který bude možné specifikovat v rámci konfigurace aplikace.

FR5.2: **E-mailové upozornění v případě selhání extrakce dat**

Pokud se nepodaří extrahovat data, aplikace pošle email s chybou na e-mail uživatele, který dokument vytvořil. V e-mailu bude uveden popis chyby a odkaz na dokument, který selhal.

6. Uživatelský profil

FR6.1: **Změna údajů v uživatelském profilu**

Uživatel bude mít možnost změnit své jméno a email.

FR6.2: **Změna hesla**

Uživatel bude mít možnost změnit své heslo.

FR6.3: **Smazání účtu**

Uživatel bude mít možnost smazat svůj účet. V případě smazání účtu se smažou i všechny jeho projekty a dokumenty.

7. Administrace

FR7.1: Správa uživatelů

Administrátor bude mít možnost spravovat uživatele, mazat jejich účty a měnit jejich role. Zároveň umožní provádět operace nad jejich projekty a dokumenty, jako je editace a mazání.

3.2.2 Nefunkční požadavky

NR1: Webová aplikace

Aplikace bude dostupná jako webová aplikace napsaná s pomocí frameworku Next.js. Uživatelé budou moci přistupovat k aplikaci pomocí webového prohlížeče.

NR2: Open-source

Aplikace bude open-source, zdrojový kód bude dostupný v git repozitáři.

NR3: Aplikace nezahltí scrapované webové stránky

Aplikace bude cachovat stahovaná data ze scrapovaných webových stránek a znovupoužívat uložené odpovědi, aby nedocházelo k zahlcení cílového webu. V rámci konfigurace aplikace bude možné nastavit, jak minimálně dlouho budou data cachována.

3.3 Modelování případů užití

V této kapitole je provedeno modelování případů užití. Nejprve jsou specifikováni aktéři, kteří budou aplikaci používat. Poté jsou uvedeny případy užití, které jsou rozděleny do několika skupin podle funkcí, které aplikace má splňovat.

3.3.1 Aktéři

V této kapitole jsou uvedeni aktéři, rozdělení podle rolí. Přehled aktérů a vazby mezi nimi je zobrazen na obrázku 3.1.

System – zajišťuje běh aplikace, zpracování dat a přístup skrze API.

Administrátoři – správa platformy, dohled nad bezpečnostními a provozními aspekty.

Požadavky: Nástroje pro správu uživatelských účtů a konfiguraci systému.

Neregistrovaní uživatelé – omezený přístup k určitým funkcím nebo datům.

Požadavky: Základní informace o platformě, možnost registrace, sandbox pro vyzkoušení extrakce.

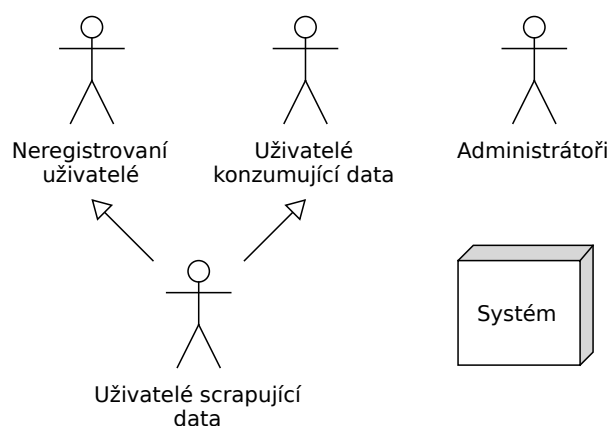
3. ANALÝZA

Uživatelé scrapující data – zadávání URL pro scraping a definice transformačního schématu.

Požadavky: Intuitivní UI pro zadávání URL, specifikace transformačního schématu, náhled extrahovaných dat, možnosti exportu.

Uživatelé konzumující data – přístup a používání výstupních dat skrze API endpoint. Mohou využít transformace definované jinými uživateli.

Požadavky: Možnost získat přístup k extrahovaným datům.



Obrázek 3.1: Diagram aktérů

3.3.2 Případy užití

V této kapitole jsou uvedeny případy užití, rozdělené do několika skupin podle funkcí, které aplikace má splňovat. Přehled vazeb mezi use-case a aktéry je zobrazen na obrázku 3.2.

UC1: Autentizace a autorizace uživatelů

UC1.1: Registrace

Aktéři: Neregistrovaný uživatel

Cíl: Zaregistrovat se do aplikace

Podmínky: Uživatel není registrovaný

Popis: Uživatel zadá přihlašovací údaje (email, heslo a kontrolu hesla) a odešle registraci. Aplikace po úspěšné registraci přeměruje na stránku s přihlášením.

UC1.2: Přihlášení

Aktéři: Administrátoři, uživatelé scrapující data

Cíl: Přihlásit se do aplikace

Podmínky: Uživatel je registrovaný

Popis: Uživatel zadá přihlašovací údaje (email a heslo) a potvrdí přihlášení. Po přihlášení je přesměrovaný na úvodní stránku.

UC1.3: Odhlášení

Aktéři: Administrátoři, uživatelé scrapující data

Cíl: Odhlásit se z aplikace

Podmínky: Uživatel je přihlášený

Popis: Uživatel potvrdí odhlášení.

UC2: Správa projektů

UC2.1: Vytvoření projektu

Aktéři: Uživatelé scrapující data

Cíl: Vytvořit nový projekt

Podmínky: Uživatel je přihlášený

Popis: Uživatel zobrazí stránku s projekty a klikne na tlačítko pro přidání projektu. Systém zobrazí dialog pro vytvoření projektu, v rámci kterého je možné zadat název projektu. Po potvrzení vytvoření projektu je obnovený seznam projektů na stránce s projekty.

Alter. scénář: Uživatel zadá projekt s názvem, který již existuje. Systém zobrazí chybu.

UC2.2: Editace projektu

Aktéři: Uživatelé scrapující data

Cíl: Editovat projekt

Podmínky: Uživatel je přihlášený, projekt existuje

Popis: Uživatel na detailu projektu klikne na tlačítko pro editaci projektu. Systém zobrazí dialog pro editaci projektu, v rámci kterého je možné změnit název projektu. Je možné přidat nebo odebrat access tokeny. Po potvrzení editace projektu je obnovený detail projektu.

Alter. scénář: Uživatel zadá projekt s názvem, který již existuje. Systém zobrazí chybu.

UC2.3: Smazání projektu

Aktéři: Uživatelé scrapující data

Cíl: Smazat projekt

Podmínky: Uživatel je přihlášený, projekt existuje

Popis: Uživatel na stránce všech projektů a klikne na tlačítko pro smazání projektu. Systém zobrazí dialog pro potvrzení smazání projektu. Uživatel potvrdí kliknutím na tlačítko a sys-

tém smaže projekt a všechny jeho dokumenty. Seznam projektů je obnovený a projekt zmizí ze seznamu.

Alter. scénář: Uživatel zruší smazání projektu.

UC3: Správa dokumentů

UC3.1: Přidání dokumentu do projektu

Aktéři: Uživatelé scrapující data

Cíl: Přidat dokument do projektu

Podmínky: Uživatel je přihlášený, projekt existuje

Popis: Uživatel na detailu projektu klikne na tlačítko přidání dokumentu. Systém zobrazí dialog pro přidání dokumentu, v rámci kterého je možné zadat URL pro extrakci dat. Zároveň je možné specifikovat pokročilé možnosti a nastavení transformace zadaného dokumentu (způsob stažení HTML stránky, cache atd.). Po potvrzení přidání dokumentu je obnovený seznam dokumentů na detailu projektů.

Uživateli je zobrazena informace o tom, zda-li zadaná doména blokuje přístup robotů skrze robots.txt soubor. Pokud ano, je zobrazeno varování. Pokud se nepodaří načíst robots.txt soubor, je zobrazeno upozornění.

Alter. scénář: Uživatel zadá dokument s názvem, který již existuje. Systém zobrazí chybu.

UC3.2: Editace dokumentu

Aktéři: Uživatelé scrapující data

Cíl: Editovat dokument

Podmínky: Uživatel je přihlášený, projekt existuje, dokument existuje

Popis: Uživatel na detailu projektu klikne na tlačítko pro editaci dokumentu. Systém zobrazí dialog pro editaci dokumentu, v rámci kterého je možné upravit parametry stejné jako při přidání dokumentu (chování odpovídá UC3.1). Po potvrzení editace dokumentu je obnovený seznam dokumentů na detailu projektů.

Alter. scénář: Uživatel zadá dokument s názvem, který již existuje. Systém zobrazí chybu.

UC3.3: Smazání dokumentu

Aktéři: Uživatelé scrapující data

Cíl: Smazat dokument

Podmínky: Uživatel je přihlášený, projekt existuje, dokument existuje

Popis: Uživatel na detailu projektu klikne na tlačítko pro smazání dokumentu. Systém zobrazí dialog pro potvrzení smazání dokumentu. Uživatel potvrdí kliknutím na tlačítko a systém smaže

dokument. Seznam dokumentů je obnovený a dokument zmizí ze seznamu.

Alter. scénář: Uživatel zruší smazání dokumentu.

UC4: Transformace pomocí UI

UC4.1: Úprava transformačního schématu

Aktéři: Uživatelé scrapující data

Cíl: Upravit transformační schéma

Podmínky: Uživatel je přihlášený, projekt existuje, dokument existuje

Popis: Uživatel v seznamu dokumentů klikne na dokument, který chce upravit. Systém přesměruje uživatele na stránku pro úpravu transformačního schématu, v rámci kterého je možné specifikovat transformační schéma prostřednictvím UI.

Alter. scénář: Uživatel zadá neplatné transformační schéma. Systém zobrazí chybu.

UC4.2: Náhled extrahovaných dat

Aktéři: Uživatelé scrapující data

Cíl: Zobrazit náhled extrahovaných dat

Podmínky: Uživatel je přihlášený, projekt existuje, dokument existuje a má validní transformační schéma

Popis: Uživatel upraví transformační schéma (odpovídá UC4.1). Při každé změně schématu je zobrazený náhled extrahovaných dat na stejné stránce jako je formulář pro editaci schématu. Náhled je zobrazený v podobě JSON.

Alter. scénář: Uživatel zadá neplatné transformační schéma. Systém zobrazí chybu popisující zadaný problém s transformací (ať už se jedná o problém v transformačním schématu, nebo v transformovaných datech).

UC5: Správa access tokenů

UC5.1: Vytvoření nového access tokenu

Aktéři: Uživatelé scrapující data

Cíl: Vytvořit nový access token

Podmínky: Uživatel je přihlášený, projekt existuje

Popis: Uživatel na detailu projektu klikne na tlačítko pro zobrazení nastavení projektu. Následně se zobrazí modální dialog, kde je možné kliknout na tlačítko pro přidání access tokenu. Systém zobrazí dialog pro vytvoření access tokenu, v rámci kterého je možné zadat popisek tokenu, expiraci tokenu a povolené origins. Po potvrzení vytvoření access tokenu se zavře dialog

3. ANALÝZA

s detailem access tokenu a obnoví se seznam tokenů v dialogu s nastavením projektu.

UC5.2: **Editace access tokenu**

Aktéři: Uživatelé scrapující data

Cíl: Editovat access token

Podmínky: Uživatel je přihlášený, projekt existuje, access token existuje

Popis: Uživatel na detailu projektu klikne na tlačítko pro zobrazení nastavení projektu. Systém zobrazí dialog s nastavením projektu, kde je i seznam access tokenů. U každého tokenu lze kliknout na tlačítko pro editaci. Systém zobrazí dialog pro editaci access tokenu. Uživatel upraví požadované parametry (popisek tokenu a nastavení *origins*) a potvrdí editaci.

UC5.3: **Smazání access tokenu**

Aktéři: Uživatelé scrapující data

Cíl: Smazat access token

Podmínky: Uživatel je přihlášený, projekt existuje, access token existuje

Popis: Uživatel zobrazí detail tokenu (postup jako UC4.2). V detailu tokenu klikne na tlačítko smazat. Systém zobrazí dialog pro potvrzení smazání tokenu. Uživatel potvrdí smazání tokenu.

Alter. scénář: Uživatel při zobrazení potvrzovacího dialogu zruší smazání tokenu, potvrzení se zavře, token zůstává nezměněný.

UC5.4: **Omezení počtu requestů pro access token**

Aktéři: Uživatelé scrapující data

Cíl: Omezit počet requestů pro access token

Podmínky: Uživatel je přihlášený, projekt existuje, dokument existuje, uživatel má k dispozici platný access token

Popis: Uživatel zobrazí dialog s nastavením projektu (jako v UC2.2). V seznamu access tokenů klikne na tlačítko nastavení limitů pro požadovaný token. Systém zobrazí dialog, kde uživatel specifikuje maximální počet požadavků a časový interval. Po potvrzení nastavení se zavře dialog s nastavením limitů a zobrazí se potvrzení o nastavení limitů.

UC6: **API endpoint**

UC6.1: **Přístup k datům přes API endpoint**

Aktéři: Uživatelé konzumující data

Cíl: Získat přístup k datům skrze API endpoint

Podmínky: Uživatel je přihlášený, projekt existuje, dokument existuje, uživatel má k dispozici platný access token

Popis: Uživatel zobrazí detail projektu, kde je vypsaný seznam dokumentů. U každého dokumentu je k dispozici tlačítko, pro zobrazení informací k API. Uživatel klikne na tlačítko API pro požadovaný dokument. Systém zobrazí modální dialog s URL adresou API endpointu, skrze který lze přistupovat k extrahovaným datům. Po zadání access tokenu se upraví vygenerované URL. Uživatel klikne na tlačítko pro otevření odkazu a v prohlížeči je v novém okně otevřená stránka s extrahovanými daty.

Alter. scénář: Uživatel má URL pro API endpoint k dispozici (buď zkopírovaný, anebo ho obdržel od jiného uživatele).

Alter. scénář: Uživatel (zde v roli vývojářů nebo integrátorů) použije pro přístup k datům jinou cestu než skrze prohlížeč (např. v rámci vlastní aplikace).

UC6.2: Zablokování další extrakce dat

Aktéři: Systém, uživatelé konzumující data

Cíl: Zablokovat další extrakci dat

Podmínky: extrakce dat selhala a uplynul minimální časový limit

Popis: V případě, že došlo k opakovanému pokusu o extrakci dat skrze přístup přes API endpoint, která skončila chybou, je zablokována další extrakce dat. Musí být splněna podmínka, že uplynulý čas od poslední chybné extrakce je větší než minimální časový limit stanovený systémem.

Systém zároveň zkontroluje, zda-li byl uživatel již informován o chybě, a pokud ne, odešle email s chybou na email uživatele, který dokument vytvořil.

Alter. scénář: Extrakce dat proběhla v pořádku. Systém vymaže informaci o chybě a resetuje časový limit.

UC7: Správa účtu

UC7.1: Změna hesla

Aktéři: Administrátoři, uživatelé scrapující data

Cíl: Změnit heslo

Podmínky: Uživatel je přihlášený

Popis: Uživatel zadá staré heslo, nové heslo a kontrolu nového hesla a potvrdí změnu hesla. Systém zobrazí potvrzení o změně hesla.

Alter. scénář: Uživatel zadá špatné staré heslo. Systém zobrazí chybu.

UC7.2: Změna údajů v uživatelském profilu

Aktéři: Administrátoři, uživatelé scrapující data

Cíl: Změnit údaje v uživatelském profilu

Podmínky: Uživatel je přihlášený

3. ANALÝZA

Popis: Uživatel zadá nové údaje (jméno a email) a potvrdí změnu. Systém zobrazí potvrzení o změně údajů.

Alter. scénář: Uživatel zadá neplatný email. Systém zobrazí chybu.

UC7.3: Smazání účtu

Aktéři: Administrátoři, uživatelé scrapující data

Cíl: Smazat účet

Podmínky: Uživatel je přihlášený

Popis: Uživatel zobrazí stránku s profilem a otevře v menu možnost smazat účet. Systém zobrazí dialog pro potvrzení smazání účtu, v rámci kterého je zobrazeno textové políčko, do kterého je potřeba zadat kontrolní text, aby bylo možné potvrdit operaci. Toto je prevence proti nechtěnému smazání účtu, jedná se o kritickou a nenávratnou akci. Po potvrzení smazání účtu je uživatel odhlášený a přeměřován na úvodní stránku.

UC8: Správa uživatelů

UC8.1: Zobrazení seznamu uživatelů

Aktéři: Administrátoři

Cíl: Zobrazit seznam uživatelů

Podmínky: Uživatel je přihlášený

Popis: Administrátor zobrazí stránku s administrace a klikne na záložku uživatelů v menu. Systém zobrazí seznam uživatelů.

UC8.2: Zobrazení detailu uživatele

Aktéři: Administrátoři

Cíl: Zobrazit detail uživatele

Podmínky: Uživatel je přihlášený, uživatel existuje

Popis: Na seznamu uživatelů klikne na požadovaného uživatele. Systém zobrazí detail uživatele s možností editace a smazání uživatele a zobrazení jeho projektů.

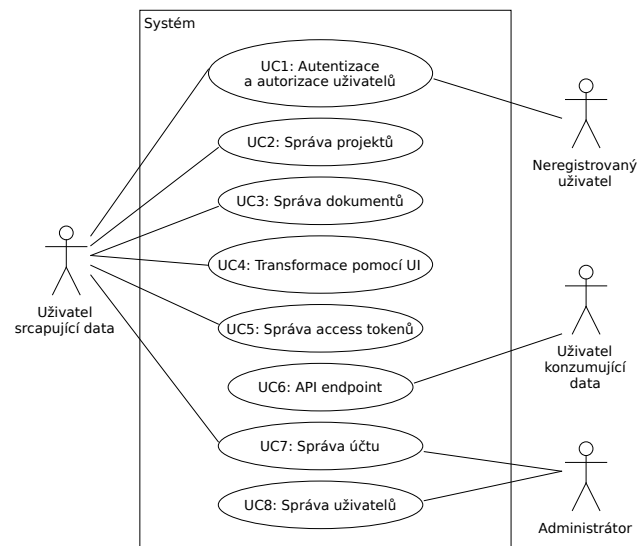
UC8.3: Nastavení limitů uživatele pro přístup k API

Aktéři: Administrátoři

Cíl: Nastavit limity uživatele pro přístup k API

Podmínky: Uživatel je přihlášený, uživatel existuje

Popis: Na seznamu uživatelů klikne na požadovaného uživatele. Systém zobrazí detail uživatele. Administrátor klikne na tlačítko nastavení limitů. Systém zobrazí dialog pro nastavení limitů, kde je možné nastavit maximální počet requestů a časový interval. Po potvrzení nastavení se zavře dialog s nastavením limitů a zobrazí se potvrzení o nastavení limitů.



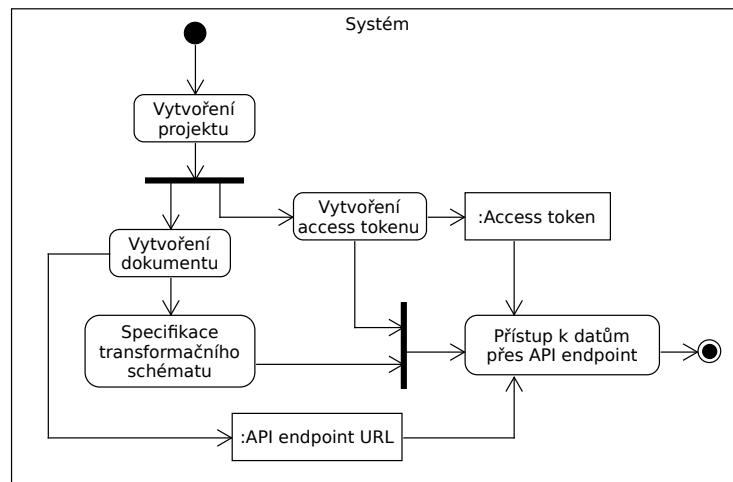
Obrázek 3.2: Přehled vazeb mezi use-case a aktéry

Funkční požadavek (FR) / Use-case (UC)	FR1.1	FR2.1	FR2.2	FR3.1	FR3.2	FR3.3	FR3.4	FR3.5	FR3.6	FR4.1	FR4.2	FR4.3	FR4.4	FR5.1	FR5.2	FR6.1	FR6.2	FR6.3	FR7.1
UC1.1: Registrace	✓																		
UC1.2: Přihlášení	✓																		
UC1.3: Odhlášení	✓																		
UC2.1: Vytvoření projektu		✓																	
UC2.2: Editace projektu		✓																	
UC2.3: Smazání projektu		✓																	
UC3.1: Přidání dokumentu do projektu			✓	✓			✓	✓	✓										
UC3.2: Editace dokumentu			✓	✓			✓	✓	✓										
UC3.3: Smazání dokumentu			✓																
UC4.1: Úprava transformačního schématu					✓														
UC4.2: Náhled extrahovaných dat						✓						✓							
UC5.1: Vytvoření nového access tokenu										✓									
UC5.2: Editace access tokenu										✓									
UC5.3: Smazání access tokenu										✓									
UC5.4: Omezení requestů pro access token													✓						
UC6.1: Přístup k datům přes API endpoint											✓	✓	✓						
UC6.2: Zablokování další extrakce dat														✓	✓				
UC7.1: Změna hesla																		✓	
UC7.2: Změna údajů v uživatelském profilu																✓			
UC7.3: Smazání účtu																			✓
UC8.1: Zobrazení seznamu uživatelů																			✓
UC8.2: Zobrazení detailu uživatele																			✓
UC8.3: Nastavení limitů uživatele k API																			✓

Tabulka 3.1: Pokrytí funkčních požadavků případy užití

3.4 Modelování procesů

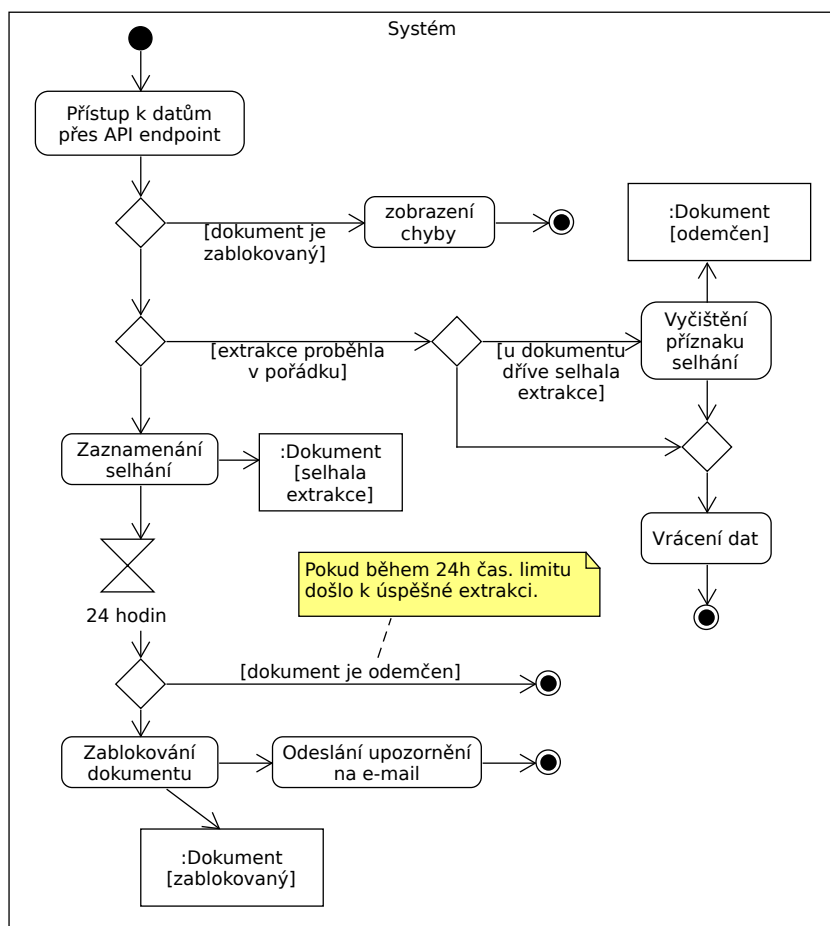
V rámci aplikace je hlavním procesem vytvoření transformačního schématu a následný přístup skrze API endpoint. Uživatel vytvoří projekt (pro ten specifikuje access token), v rámci něj přidá dokument (tím se vygeneruje ID dokumentu a současně URL pro API endpoint), v rámci kterého specifikuje transformační schéma. Následně může přistupovat k datům skrze API endpoint. Proces je znázorněn na obrázku 3.3.



Obrázek 3.3: Proces vytvoření transformace a přístupu skrze API endpoint

Během přístupu skrze API endpoint může dojít k selhání transformace. Příčin selhání může být více – buď dojde k chybě při stahování dat (např. je cílový server dočasně nedostupný), anebo dojde k chybě během samotné transformace (např. uzel není nalezen v HTML dokumentu). Pokud k takové chybě dojde, aplikace tento stav zaznamená a začne běžet 24-hodinový časový limit (zde pevně stanoveno pro ukázkou, nicméně tento limit bude možné specifikovat v rámci konfigurace aplikace). Pokud během tohoto limitu dojde k úspěšné extrakci, příznak selhání se vyčistí a transformace se považuje za odblokovanou. Pokud však uběhne 24hodinový limit a nebylo během něj dosaženo úspěšné extrakce, je transformace zablokována a uživatel musí zasáhnout. (Opravit chybu extrakce, např. změna URL anebo úprava transformačního schématu, a následně uložení provedených změn, čímž se dokument odblokuje) Zablokování transformace je znázorněno na obrázku 3.4.

3. ANALÝZA



Obrázek 3.4: Proces zablokování transformace při přístupu skrze API

Návrh

V této kapitole je popsán návrh aplikace. Nejprve se zaměříme na seznam technologií a knihoven, které byly použity při vývoji aplikace. Následně je popsána architektura aplikace, která popisuje rozdělení na hlavní aplikaci *MakeAPI* a službu *URL Fetcher*. Následně je popsán diagram nasazení a databázový model. Po té je popsán algoritmus pro transformaci dat a reprezentace struktury transformačního schématu. Na závěr je popsán rate-limiting pro API endpoint.

4.1 Použité technologie

Pro vývoj webové aplikace jsme zvolili následující stack technologií a knihoven. (Volba byla ovlivněna předchozími zkušenostmi autora s danými technologiemi. Při tvorbě aplikace bylo použito více knihoven, následující seznam obsahuje pouze ty nejdůležitější.)

TypeScript	rozšíření jazyka JavaScript o statické typování
Node.js	JavaScriptové prostředí pro běh aplikace na serveru
Next.js	framework pro tvorbu full-stack React.js aplikací
NextAuth.js	knihovna pro autentizaci uživatelů
React.js	knihovna pro tvorbu client-side UI
PostgreSQL	relační databáze
Prisma	ORM pro Node.js a TypeScript
Express.js	knihovna pro tvorbu API serveru
Redis	key-value databáze, vhodná pro cachování dat
Puppeteer	knihovna pro Node.js, umožňující procházet webové stránky a manipulovat s nimi skrze headless prohlížeč
Axios	knihovna pro vytváření HTTP requestů

Cheerio knihovna pro parsování HTML
Zod knihovna pro validaci schématu dat pro TypeScript

4.2 Architektura aplikace

Aplikace je navržena jako webová aplikace, rozdělená na několik částí. Hlavní aplikace – *MakeAPI* –, zajišťující provoz webu, správu uživatelů, správu projektů a dokumentů a zpřístupnění API endpointu, je napsaná s pomocí Next.js frameworku. Pro vykreslení UI používá knihovnu React.js.

Pro stahování dat ze zadaných URL je navržena samostatná služba – *URL Fetcher* –, která zajišťuje stahování dat prostřednictvím *raw requestů* (jednoduchý HTTP request) anebo *headless requestů* (využívá prohlížeč pro stažení dat). Tato komponenta poskytuje přístup skrze HTTP API. V rámci naší aplikace zastupuje roli proxy serveru.

Architektura je hybridní, využívá microservice architekturu pro propojení hlavní aplikace *MakeApi* se službou *URL Fetcher*, jednotlivé aplikace pak využívají monolitickou architekturu.

4.2.1 Aplikace MakeAPI

Hlavní část aplikace je napsána v TypeScriptu pomocí Next.js frameworku¹¹. Dle dokumentace Next.js [36] se jedná full-stack framework pro React.js. Pro tvorbu UI se využívá knihovna React.js. Next.js pak zajišťuje zejména rendering, routing, data-fetching a optimalizace.

Pro ukládání dat je využita databáze PostgreSQL¹² společně s knihovnou Prisma¹³, která zajišťuje snadnou integraci s aplikací skrze ORM.

4.2.1.1 Komponenty

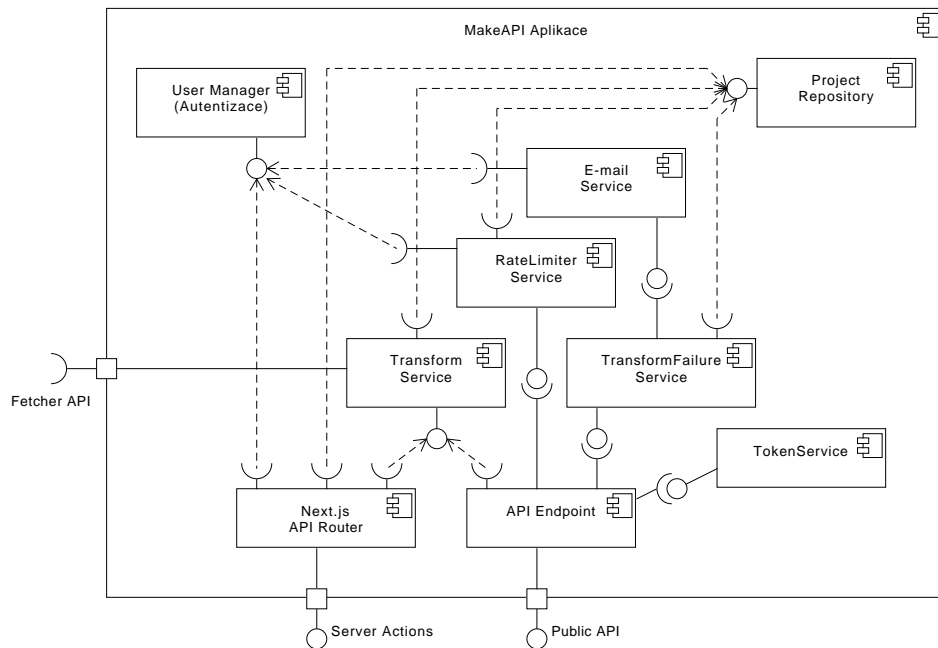
Hlavní aplikace se skládá z několika komponent. Jejich přehled je zobrazen na diagramu na obrázku 4.1. (Pro zjednodušení jsou některé komponenty seskupeny do jedné, ve výsledné implementaci budou rozdělené na více částí.) Jedná se o následující komponenty:

User Manager – zajišťuje autentizaci a zpřístupnění informací o uživateli (jméno, e-mail, role). Využívá jej komponenta *Next.js App Router* pro autorizaci a vykreslení UI na základě přihlášení a uživatelské role. Zároveň jej využívá komponenta *E-mail service* pro získání kontaktních údajů pro zadaného uživatele. Komponenta *RateLimiter Service* jej využívá pro získání informací o limitech pro uživatele.

¹¹<https://nextjs.org/>

¹²<https://www.postgresql.org/>

¹³<https://www.prisma.io/>



Obrázek 4.1: Komponenty MakeAPI aplikace

ProjectRepository – ukládá a zprostředkovává informace o uložených projektech uživatele (vč. dokumentů a uloženým transformačním schémata-tům.) Využívá jej komponenta *Next.js App Router* pro vykreslení UI, *Transform Service* pro získání transformačního schématu pro zadaný dokument, *RateLimiter Service* pro získání informací o limitech pro projekt a *TransformFailure Service* pro získání informací o selháních transformace.

TokenService – poskytuje informace o access tokenech pro projekty. Využívá jej komponenta *API Endpoint* pro autorizaci přístupu k transformačním skrze veřejné API.

E-mail Service – zajišťuje odesílání e-mailů. Využívá jej komponenta *TransformFailure Service* pro odesílání upozornění na e-mail při selhání transformace.

TransformFailure Service – zajišťuje ukládání informací o selháních transformace. Využívá jej komponenta *API Endpoint* pro získání informací o selháních transformace.

Ratelimiter Service – zajišťuje ukládání informací o přístupech uživatele k veřejnému API endpointu. Využívá jej komponenta *API Endpoint* pro aplikaci rate-limitingu requestů.

Transform Service – zajišťuje transformaci dat. Využívá jej komponenta *API Endpoint* pro transformaci dat skrze veřejné API a komponenta *Next.js App Router* pro transformaci dat skrze UI. Pro získání HTML dokumentu vyžaduje rozhraní *Fetcher API* (v diagramu reprezentuje závislost na externí službě *URL Fetcher*, která zajišťuje stahování dat ze zadaných URL.)

Next.js App Router – reprezentuje součást Next.js frameworku, která zajišťuje vykreslení UI skrze *App Routes* a přístup k volání serverových akcí skrze *Server Actions*.

4.2.2 Služba URL Fetcher

URL Fetcher je navržen jako samostatná služba, která je napsána pro Node.js s pomocí Express.js knihovny. Služba poskytuje dva typy stahování dat.

První typ je *raw request*, který využívá knihovnu Axios¹⁴ pro stahování dat. Druhý typ je *headless request*, který využívá knihovnu Puppeteer¹⁵ pro načtení stránky v prohlížeči a následné stažení dat.

Pro cachování dat je využita databáze *Redis* [37]. Redis je key-value databáze, která ukládá data do paměti. Komponenta ji využívá pro cachování stažených dat. Pro organizaci cachovaných requestů je využit požadovaný *URL* jako klíč.¹⁶ Hodnotou je pak objekt obsahující obsah HTTP response, jeho *content-type*, *max-age* a *timestamp*. (Tyto parametry pak slouží k rozhodnutí o použití cachovaných dat při následných requestech.)

4.3 Diagram nasazení

Diagram nasazení je zobrazen na obrázku 4.2. Zobrazuje komponenty aplikace *MakeAPI*, rozdělené na dvě části – **MakeAPI Frontend**, která běží na straně uživatele v prohlížeči a zobrazuje UI, a **MakeAPI Server**, která běží na straně serveru v prostředí Node.js a zajišťuje operace backendu. Komunikace mezi nimi probíhá skrze *Server Actions*¹⁷ a *Next.js App Routes*¹⁸. Pro perzistenci dat využívá databázi PostgreSQL skrze ORM Prisma.

¹⁴<https://axios-http.com/>

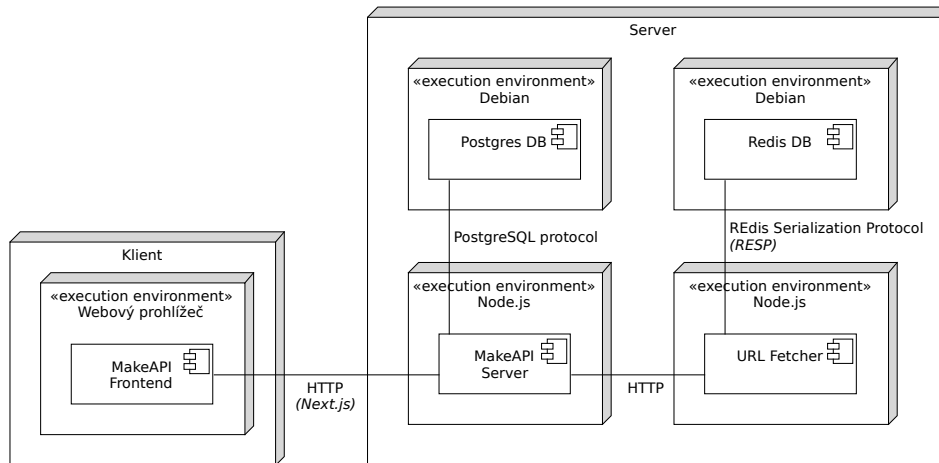
¹⁵<https://pptr.dev/>

¹⁶Pro *raw* requesty využívá namespace `raw-content-cache` a pro *headless* requesty namespace `html-content-cache`.

¹⁷Jedná se o HTTP požadavky zapouzdřené frameworkem Next.js

¹⁸Zobrazení stránek na základě URL cesty

MakeAPI Server komunikuje se službou **URL Fetcher** skrze HTTP požadavky. Tato služba běží v prostředí Node.js. Pro cachování stažených dat využívá databázi Redis. Služba je oddělená od hlavní aplikace zejména z důvodu **škálování**, je možné nasadit více instancí této služby, aby byla rozprostřena její zátěž. Hlavním důvodem je zejména náročnost zpracování *headless* requestů, které vyžadují delší čas.



Obrázek 4.2: Diagram nasazení aplikace MakeAPI

4.4 Databázový model

Databázový model aplikace se skládá z následujících entit. E-R diagram je zobrazen na obrázku 4.3.

User – informace o registrovaných uživateli, obsahuje také roli, která je buď **USER** pro běžné uživatele, anebo **ADMIN** pro administrátory

Project – seznam projektů pro uživatele, projekt může mít přiřazených více dokumentů (*ProjectDocument*) a access tokenů (*ProjectAccessToken*)

ProjectDocument – dokument v projektu, který obsahuje URL cílového dokumentu, informace způsobu stahování (*downloadStrategy* je buď **RAW** anebo **HEADLESS**), nastavení *max-age*, flag pro nastavení striktní transformace (*strictTransform*; pokud je **true**, pak jsou všechny chyby během transformování vyhodnoceny jako selhání transformace)

TransformationSchema – definice transformačního schématu dokumentu

TransformFailure – udržuje informace o chybě během transformace (při přístupu skrze API endpoint), obsahuje timestamp prvního a posledního selhání, informaci, zda-li bylo odesláno upozornění na e-mail a timestamp zablokování transformace

ProjectAccessToken – access token pro přístup k API endpointu, obsahuje také seznam *origins*, které mají povolený přístup k API endpointu skrze tento token

RateLimit – nastavení rate-limitingu (pro uživatele nebo access token), obsahuje počet requestů za zadané časové okno

4.5 Transformace dat

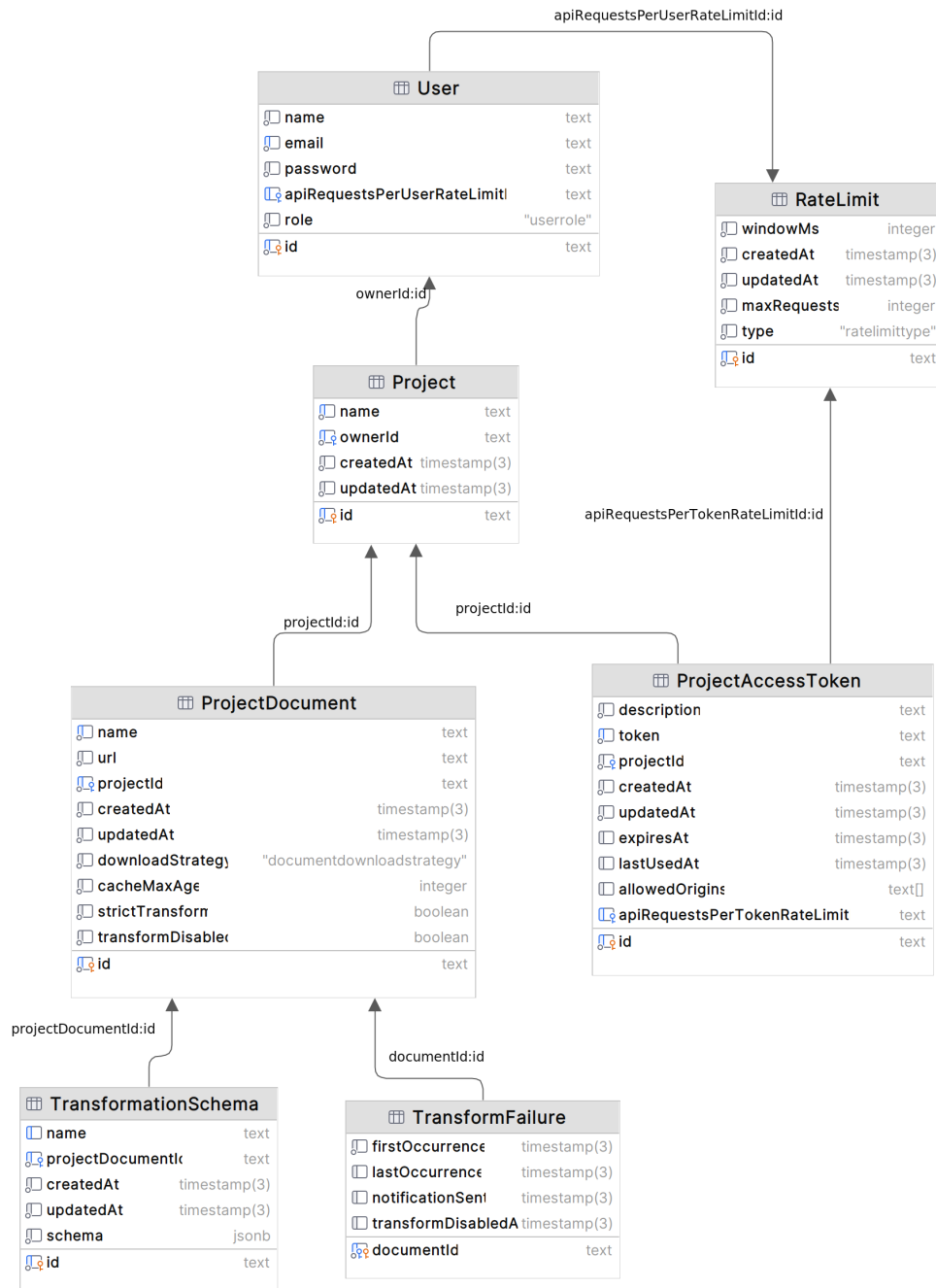
Transformace dat z HTML formátu do JSON je stěžejní funkcionalita aplikace. Transformace je navržena jako rekurzivní algoritmus, který prochází stromovou strukturu transformačního schématu a na základě selektorů vybírá data z HTML dokumentu. Uživatel při vytváření schématu přímo určuje výstupní strukturu dat.

Pro reprezentaci schématu jsme využili *composite pattern*[38]. Každý uzel schématu je reprezentován objektem **NodeType**, který obsahuje informaci o typu uzlu (objekt, pole, text atd.) a případně další informace (např. selektor, kterým se vybírají data z HTML dokumentu). Zároveň každý uzel obsahuje funkci **transform**, která na základě parametrů provede transformaci vstupních dat na zadaném podstromu HTML dokumentu (na diagramu popsáno jako obecný *HTMLElement* uzel, avšak v samotné implementaci bude využita knihovna Cheerio pro traversování HTML dokumentu). Struktura reprezentace transformačního schématu je znázorněna na obrázku 4.4

Funkce **transform** dostává na vstupu i informaci o aktuální cestě ve struktuře transformačního schématu a referenci na objekt **context**, který umožňuje jednotlivým uzlům zaznamenat informaci o chybě (pomocí **addError** metody) a také zpřístupňuje instanci CheerioAPI pro vstupní HTML soubor, který mohou uzly využít pro snadnější přístup k prvkům v dokumentu. Context je sdílený napříč celým stromem a je vytvořen před samotnou transformací v rámci *TransformService* (viz obrázek 4.6).

Context je využíván pro zobrazování chyb během transformace při přístupu skrze API endpoint. Využitý je také v rámci UI pro zobrazování chyb uživateli během definování transformačního schématu.

Pro vykreslení jednotlivých uzlů v rámci formuláře jsou využity UI komponenty **TransformationNode**. Definují vlastnosti (typ, selector, ...) uzlu příslušného typu (viz tabulka 4.1). Diagram tříd je zobrazen na obrázku 4.5.

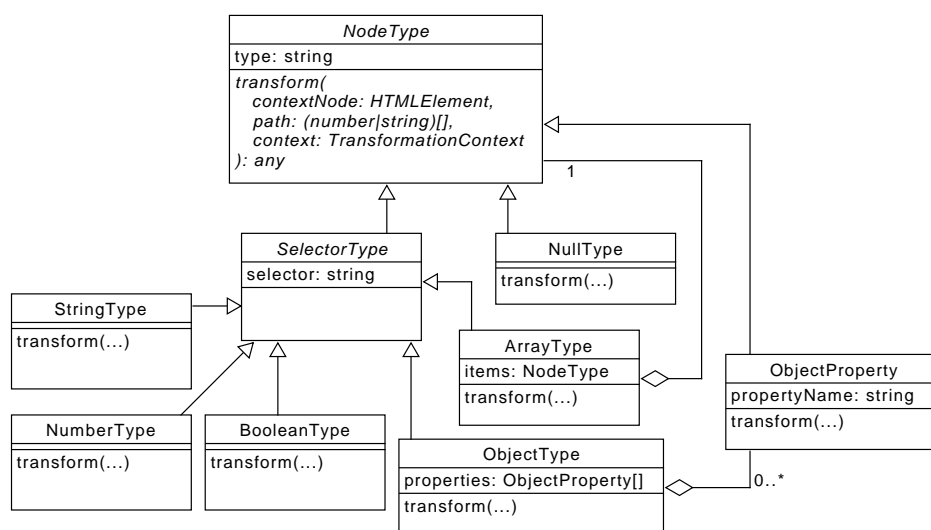


Obrázek 4.3: Databázový model aplikace

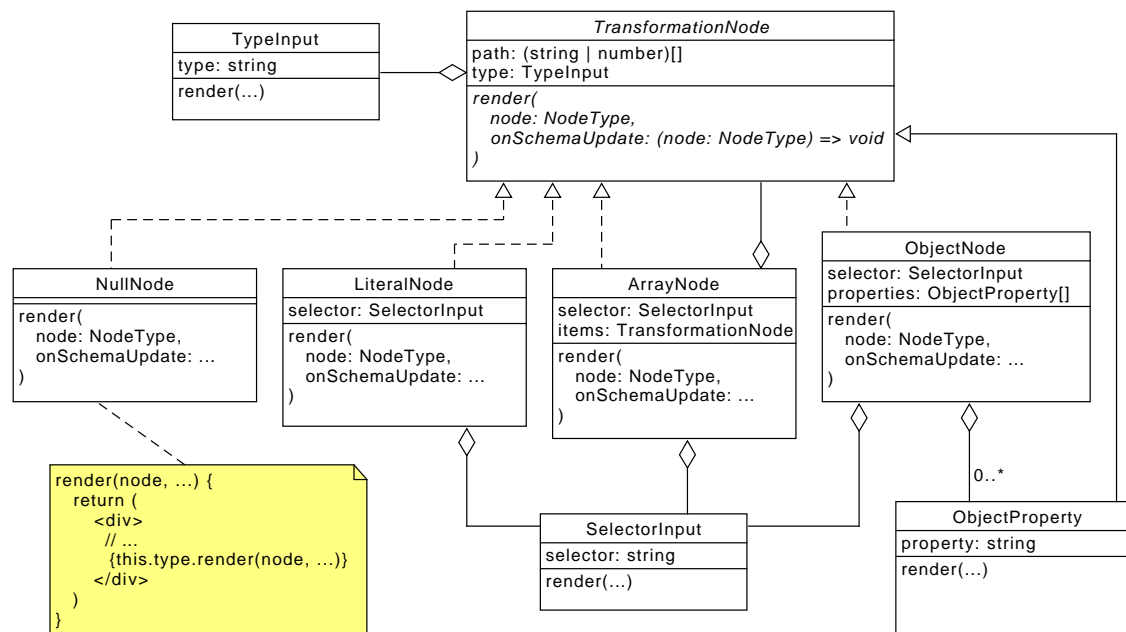
4. NÁVRH

NodeType	TransformationNode (UI)
NullType	NullNode
StringType	LiteralNode
NumberType	LiteralNode
BooleanType	LiteralNode
ArrayType	ArrayNode
ObjectType	ObjectNode

Tabulka 4.1: Tabulka typů uzlů a jejich příslušná komponenta ve formuláři



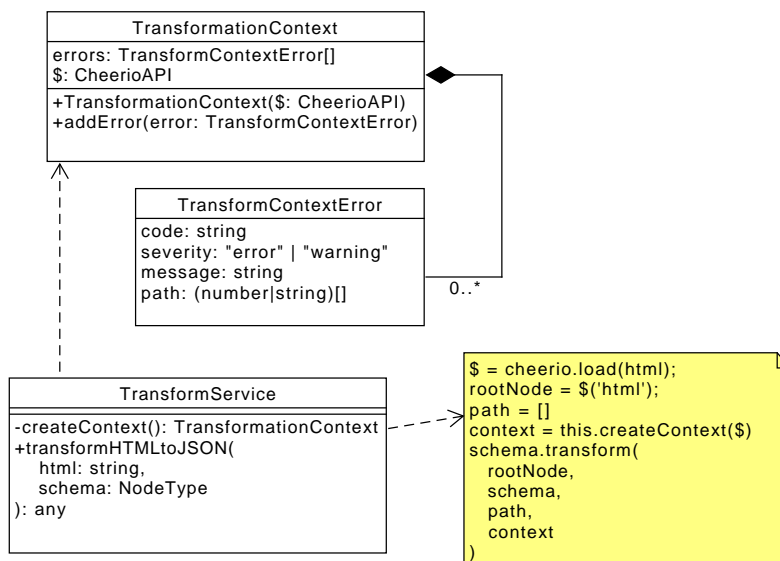
Obrázek 4.4: Reprezentace transformačního schématu s využitím composite patternu



```

render(node, ...) {
  return (
    <div>
      // ...
      {this.type.render(node, ...)}
    </div>
  )
}
  
```

Obrázek 4.5: UI komponenty transformace

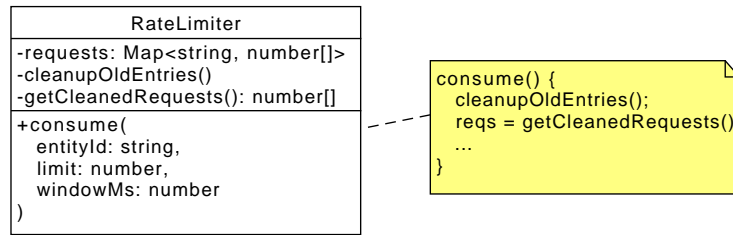


Obrázek 4.6: TransformService s context objektem pro zaznamenávání chyb během transformace

4.6 Rate-limiting

Pro rate-limiting je využit algoritmus *Sliding window log* [39]. Využívá cache pro zaznamenávání času requestu pro zadanou *entitu*¹⁹ (metoda `consume`). Jakmile přijde nový request, jsou z cache odstraněny všechny záznamy starší, než je aktuální časové okno (metoda `getCleanedRequests`) a je zkontrolováno, zda počet requestů v aktuálním okně nepřesahuje limit. V opačném případě je zaznamenán timestamp aktuálního requestu. Aby se nehromadily záznamy pro dlouho nepoužívané entity, je jednou za čas vymazána cache (pomocí metody `cleanupOldEntries`) Objekt je znázorněn na obrázku 4.7.

¹⁹Entita zde může být obecně cokoliv, v našem případě však budeme limitovat podle uživatele a podle access tokenu, název entity pak bude mít strukturu `user:<userId>` a `access-token:<tokenId>`



Obrázek 4.7: RateLimiter objekt pro zaznamenávání přístupů

Implementace prototypu

V této kapitole je popsána implementace prototypu. Ten je rozdělený na dvě části – hlavní *aplikace MakeAPI* a služba *URL Fetcher*. Nejprve je rozebrána struktura aplikace, routing (jednotlivé stránky) a middleware (zajišťující kontrolu práv k přístupu jednotlivých stránek). Dále jsou popsány jednotlivé podslužby: *autentizace a autorizace uživatelů*, *odesílání e-mailů*, *rate-limiting*, *repository* (DB entity), *selhání extrakce*, *stahování obsahu a transformace dat*. Následně je popsáno uživatelské rozhraní. V poslední části je popsána služba *URL Fetcher*, což je *proxy*, která zajišťuje stahování dat požadovaných stránek z webu.

5.1 Zdrojový kód

Codebase celé aplikace je dostupný jako open-source na veřejném git repozitáři. Kód hlavní *aplikace MakeAPI* je oddělen od služby *URL Fetcher*. Dostupné jsou na adresách:

Aplikace MakeAPI

<https://gitlab.fit.cvut.cz/vanisond/make-api-fe>

URL Fetcher

<https://gitlab.fit.cvut.cz/vanisond/make-api-be-fetcher>

5.2 Struktura aplikace MakeAPI

Struktura souborů hlavní aplikace *MakeAPI* se řídí konvencemi²⁰ pro Next.js projekt. Přehled souborů a adresářů je uveden v následujícím stromu:

²⁰<https://nextjs.org/docs/getting-started/project-structure>

```

make-api-fe
├── app/ .....komponenty Next.js App Routeru
├── components/ .....React.js komponenty
│   ├── auth/ ..... přihlašovací/registrační formulář
│   ├── navigation/ .....společné komponenty navigace
│   ├── transformation/ .....komponenty editoru pro transformaci dat
│   └── ui/ .....UI komponenty (modal dialog, záložky, tlačítka, ...)
├── lib/ ..... knihovny a pomocné funkce
│   ├── emailService/ ..... služba pro odesílání e-mailů
│   ├── rateLimitService/ .....rate-limiter
│   ├── repository/ .....přístup k DB entitám
│   ├── transformFailureService/ .zaznamenávání selhání transformace
│   ├── urlContentService/ ..... napojení na URL Fetcher službu
│   ├── auth.ts .....konfigurace pro autentizaci skrze NextAuth.js
│   └── prisma.ts .....instance PrismaClient
├── prisma/
│   └── schema.prisma ..... definice DB schématu pro Prisma ORM
├── public/ ..... veřejně přístupné soubory (obrázky apod.)
├── redux/ ..... Redux state management
├── .env .....konfigurační proměnné prostředí
├── .env.example .....šablona konfiguračních proměnných
├── middleware.ts .....Next.js middleware
└── ... ..... další konfigurační soubory projektu

```

5.2.1 Next.js App Router

Složka `app/` obsahuje React.js komponenty, které na základě jmenných konvencí přebírá *Next.js App Router* a vykresluje je podle následujících pravidel:

name Názvy složek uvnitř `/app` reprezentují URL segmenty. Např. složka `/app/projects/settings` představuje URL cestu `/projects/settings`.

page.tsx Soubor, který obsahuje komponentu, která je vykreslena na URL specifikované cestou uvnitř složky `/app`. Například soubor s cestou `/app/projects/page.tsx` představuje komponentu, která je vykreslena na URL `/projects`.

layout.tsx Soubor, který představuje *layout* komponentu, obalující všechny komponenty uvnitř složky. Komponenta dostává na vstupu defaultně parametr `children`, což odpovídá React komponentě `page.tsx`.

[name] Složka s názvem `[name]` představuje parametr URL cesty. Například složka `[projectId]` představuje parametr `projectId` v URL cestě `/projects/[projectId]`.

(name) Složka s názvem **(name)** umožňuje logické seskupování komponent podle stejné kategorie/skupiny. Tato složka neovlivňuje URL cestu. Například soubory `/app/(auth)/login/page.tsx` a `/app/login/page.tsx` by tak představovaly stejnou URL cestu `/login`²¹.

@name Složka s názvem **@name** představuje jmenný **slot**, paralelní komponentu, která je předána do *layout* komponenty na stejné úrovni. Uvnitř ní je pak předána jako parametr (s názvem odpovídající názvu složky bez prefixu „@“). Například soubor `/app/@projects/page.tsx` by byl k dispozici v souboru `/app/layout.tsx` jako parametr `projects`.

5.2.2 Komponenty routeru

Struktura komponent routeru je rozdělena následovně:

```

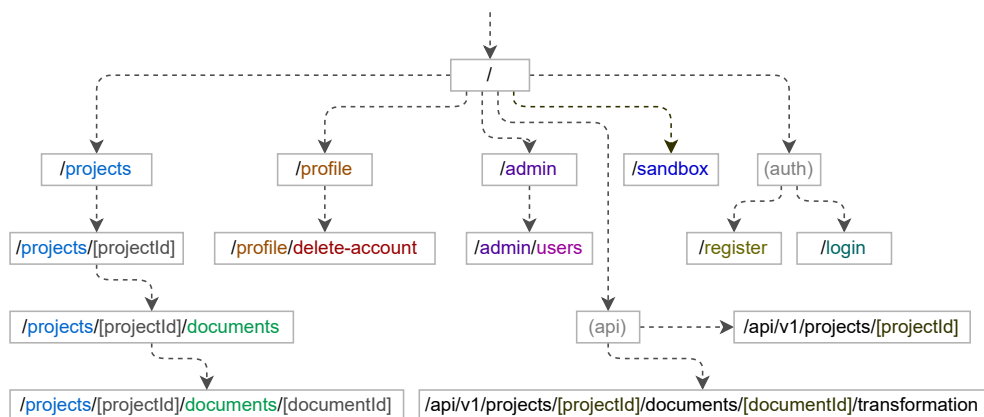
app/
├── (api)/ .....API endpointy
├── (auth)/ ..... NextAuth.js komponenty a API handler
├── (forms)/ ..... formuláře
├── admin/ ..... administrace a správa uživatelů
├── profile/ ..... editace uživatelského profilu
├── projects/ ..... správa projektů, dokumentů a transformací
└── sandbox/ ..... sandbox pro testování transformace

```

Projekt obsahuje následující *routes*:

- Route **admin** obsahuje komponenty pro administraci a správu uživatelů. Uživatele je možné zobrazit, nastavit jim limity a upravit jejich profil a přiřadit roli. Zároveň je možné zobrazit seznam všech projektů pro každého uživatele.
- Route **profile** obsahuje komponenty pro editaci uživatelského profilu. Uživatel může upravit své jméno, e-mailovou adresu a heslo. Zároveň také umožňuje smazat svůj účet se všemi daty.
- Route **projects** obsahuje komponenty pro správu projektů. Uživatel může vytvořit nový projekt a přidat do něj dokumenty. Pro každý projekt umožňuje nastavit access tokeny. Pro každý dokument umožňuje specifikovat transformační schéma.
- Route **sandbox** stránku pro testování vytváření transformačního schématu. Není možné takové transformace uložit. Jedná se zejména pro ukázkou funkčnosti editoru pro neregistrované uživatele.

²¹Next.js by však v takovém případě vyhodil chybu, neboť by nedokázal odlišit, kterou komponentu vykreslit. *Route* reprezentující danou stránku je možné specifikovat pouze jedním možným způsobem, aby nedošlo ke kolizi.



Obrázek 5.1: Struktura URL stránek aplikace MakeAPI

Pro některé komponenty je struktura dále jemněji rozčleněna do skupin (složky z názvem v kulatých závorkách):

- Skupina **(api)** obsahuje veřejné API endpointy pro **projekty**²² a **transformaci dat**²³ pro jednotlivé dokumenty v rámci projektu.
- Skupina **(auth)** obsahuje komponenty pro přihlášení a registraci uživatele a dále API handler pro NextAuth.js²⁴.
- Skupina **(forms)** obsahuje formuláře, které jsou využity v ostatních částech aplikace. Jednotlivé formuláře je možné použít na více místech, proto jsou umístěny v samostatné složce, namísto konkrétního umístění v rámci jednotlivých stránek. Obsahuje formulář pro přidání a editaci **projektu** (project), **dokumentu** (document), **transformace** (transform), **access tokenů** (access-token), nastavení limitů **rate-limiteru** (rate-limit) a **uživatelského profilu** (user). Komponenty jsou vnořeny do složky /app proto, že využívají *Server Actions*²⁵ pro zpracování a ukládání dat formulářů.

Strom souborů tvoří strukturu URL stránek, zobrazených na obrázku 5.1.

²²/api/v1/projects/[projectId]

²³/api/v1/projects/[projectId]/documents/[documentId]/transformation

²⁴<https://next-auth.js.org/configuration/initialization#route-handlers-app>

²⁵<https://nextjs.org/docs/app/building-your-application/data-fetching/server-actions-and-mutations>

5.2.3 Middleware

Pro kontrolu přístupu k jednotlivým stránkám je využit *middleware* ²⁶. Middleware je funkce v souboru `/middleware.ts`, která je vykonána před zpracováním každého požadavku na server. Kontroluje následující podmínky:

- `/` – kořenová cesta je veřejně přístupná,
- `/projects` a `/profile` – přístup pouze pro přihlášené uživatele,
- `/admin` – přístup pouze pro přihlášeného uživatele s rolí ADMIN (v opačném případě je vrácena chybová odpověď s HTTP kódem 403 – *Forbidden*),
- `/login` a `/register` – přístup pouze pro nepřihlášené uživatele. V případě přihlášeného uživatele je přesměrován na `/projects`.

5.3 Služby aplikace

V této části popíšeme jednotlivé služby, které jsou využity v rámci aplikace MakeAPI. Jedná se o služby pro autentizaci a autorizaci uživatelů, odesílání e-mailů, rate-limiting, repository (přístup k DB entitám), selhání extrakce, stahování obsahu a transformaci dat.

5.3.1 Autentizace a autorizace uživatelů

Pro autentizaci a autorizaci uživatelů je využita knihovna **NextAuth.js**. Konfigurace je v souboru `/lib/auth.ts`. Využíváme *CredentialsProvider* s vlastní implementací pro autorizaci pomocí e-mailu a hesla.

Výhodou je jednodušší implementace, avšak lepším řešením by bylo využít jiného *provideru* pro autentizaci, například *OAuth*, který by umožnil přihlášení pomocí existujícího účtu z jiné služby. Knihovna NextAuth.js poskytuje předdefinované *providery* např. pro Github, Auth0, Google aj.

5.3.2 Odesílání e-mailů

Pro odesílání e-mailů je využita služba **EmailService**. Využívá Nodemailer [40], což je knihovna pro Node.js, která umožňuje odesílat e-maily pomocí SMTP protokolu. Poskytuje také webové rozhraní pro kontrolu odeslaných e-mailů. Služba je implementována v `/lib/emailService/nodemailer.ts` a poskytuje funkci `sendMailToUser`, která přijímá parametry `userId` a `options` (odpovídá parametrům pro knihovnu *Nodemailer* – předmět, obsah, ...). Funkce získá e-mailovou adresu uživatele z DB a odešle e-mail na adresu uživatele.

²⁶<https://nextjs.org/docs/app/building-your-application/routing/middleware>

Funkce vyhodí výjimku v případě, že uživatel se zadaným id neexistuje, anebo se nepodaří e-mail odeslat.

```
const userId = "clq0yruhd00003he021a2ipb5";
const info = await sendMailTo(userId, {
  subject: "Hello!", // Subject line
  text: "Hello world?", // plain text body
  html: "<b>Hello world?</b>", // html body
});
```

Výpis kódu 5.1: Příklad odeslání e-mailu

5.3.3 Rate-limiting

Pro omezení počtu požadavků na API je využita služba **RateLimitService**. Je rozdělena na dvě části – samotný **RateLimiter** (`/lib/rateLimitService/rateLimiter.ts`), který poskytuje funkci `consume`, která přijímá parametry `entityId` (identifikace entity, pro uživatele by mohlo id mít formát např.: „user:clq0yruhd00003he021a2ipb5“) a nastavení limitů `limit` a `windowMs` (odpovídá maximálnímu povolenému počtu požadavků za zadané časové okno). Funkce vrací objekt s následujícími parametry:

- `limit` (number) – maximální povolený počet požadavků v časovém okně. (Odpovídá vstupnímu parametru `limit`),
- `remaining` (number) – počet zbývajících požadavků v aktuálním časovém okně,
- `msBeforeNext` (number) – počet milisekund, které zbývají do resetu časového okna,
- `isLimited` (boolean) – příznak, zda je aktuální požadavek omezený rate-limiterem.

Druhou částí je aplikace limitů skrze funkci `applyRateLimitPolicy` v rámci **applyApiLimits** modulu (v `/lib/rateLimitService/applyApiLimits.ts`). Funkce omezuje přístup k API endpointu, jednak pro uživatele, druhak pro access-token. Nastavení bere z DB entity `RateLimit` pro daného uživatele (parametr `apiRequestsPerUserRateLimit` entity `User`) a access-token (parametr `apiRequestsPerTokenRateLimit` entity `ProjectAccessToken`). Vrací pak objekt s informací, zda-li byl některý z limitů překročen, a hlavičky pro rate-limiting, které jsou přidány do odpovědi pro požadavek na API endpoint.

Rate-limiter je implementovaný *in-memory*, což má výhodu v rychlosti a jednoduchosti implementace. Problém však může nastat v případě velkého

množství uživatelů a access tokenů, kdy by mohlo docházet k zahlcení paměti (pro každý access token a každého uživatele se udržuje informace zvlášť).

Problém nastane v případě horizontálního škálování aplikace. Informace o překročení limitu se drží na každé instanci zvlášť a bylo by nutné synchronizovat informaci napříč instancemi.

Nevýhodou je také, že v případě restartu aplikace dojde ke ztrátě všech informací o překročení limitů. To se dá obejít nastavením rozumně nízké délky časového okna rate-limiteru (např. 1 hodina), kde by ztráta informace nehrála velkou roli.

V případě potřeby by bylo možné využít externí databázi pro synchronizaci informací, například s využitím Redis. V rámci implementace prototypu pro zajištění omezení přístupu pomocí rate-limitingu však dostačuje zvolené řešení.

5.3.4 Repository

Pro přístup k DB entitám jsou využity funkce **repository** (jednotlivé soubory v `/lib/repository/*.ts`). Využívá ORM knihovnu Prisma [41], která umožňuje přístup k DB entitám pomocí *PrismaClient* (její instance je vytvořena v `/lib/prisma.ts`). Repository poskytuje funkce pro operace nad jednotlivými entitami:

- *documents* – funkce pro operace nad entitou *Document* (získání dokumentu, seznamu dokumentů projektu, počtu dokumentů, vytvoření, editace, smazání) a *TransformFailure* (získání informace o selháních extrakce pro daný dokument, vytvoření a smazání),
- *projectAccessTokens* – funkce pro operace nad entitou *ProjectAccessToken* (získání access tokenu, seznamu access tokenů, vytvoření, editace, smazání a ověření platnosti),
- *projects* – funkce pro operace nad entitou *Project* (získání projektu, seznamu projektů pro uživatele, vytvoření, editace, smazání),
- *rateLimits* – funkce pro operace nad entitou *RateLimit* (získání limitu, vytvoření, editace, nastavení pro uživatele a access token),
- *transformationSchemas* – funkce pro operace nad entitou *TransformationSchema* (získání transformace, vytvoření, editace),
- *users* – funkce pro operace nad entitou *Users* (získání uživatele, vytvoření, editace, smazání).

Pro hashování hodnot je využita knihovna *bcrypt*²⁷, která umožňuje hashování a porovnávání hashovaných hodnot. V aplikaci se využívá se pro hashování hesel uživatelů a access tokenů.

²⁷<https://github.com/kelektiv/node.bcrypt.js>

5.3.5 Selhání extrakce

Pro zaznamenávání selhání extrakce využíváme **TransformFailureService**. Je implementována v `/lib/transformFailureService/failureManager.ts` a poskytuje *factory* metodu `createFailureReporter`, která přijímá parametry `documentId` a `reporterOptions` (doplňující informace pro reporter, současně přijímá pouze `projectId`). Vrací objekt s metodami:

- `reportTransformFailure` – metoda pro zaznamenání selhání extrakce. Přijímá parametr `reasonCode` (identifikační kód chyby, může být libovolný textový řetězec). Uloží záznam o chybě do databáze. Zároveň zkontroluje, zda-li nebyl překročen časový limit od posledního selhání. Pokud ano, zablokuje dokument a zároveň odešle e-mail s upozorněním na adresu uživatele, který dokument vlastní.
- `resetTransformFailure` – metoda pro resetování selhání extrakce. Vymaže záznam o selhání pro daný dokument z databáze.

Příklad využití je uveden v kódu 5.2.

```
// document = getProjectDocumentById(...)
const { reportTransformFailure, resetTransformFailure } =
  createFailureReporter(document.id, { /* ... */ });

// Pokud dojde k chybě, zaznamenáme selhání
if (isTransformInvalid(/* ... */)) {
  await reportTransformFailure('transform-invalid');
  return;
}

// V případě úspěchu, obnovíme případné selhání
await resetTransformFailure();
```

Výpis kódu 5.2: Příklad využití služby `TransformFailureService`

V rámci prototypu se do databáze ukládá informace pouze o poslední chybě. Nevýhodou je, že jakmile je chyba odstraněna, je záznam o selhání vymazán. Pro detailnější statistiky by bylo potřeba ukládat informaci o každé chybě (například s využitím Redis databáze). Pro účely zabránění další extrakce, v případě selhání, zvolené řešení dostačuje.

5.3.6 Stahování obsahu

Pro stahování obsahu z URL je využita služba **UrlContentService**. Je implementována v souboru `/lib/urlContentService/htmlFetcher.ts` a poskytuje funkci `fetchHtml`, která přijímá parametr `url` (zdrojová URL dokumentu)

pro stažení) a **strategy** (typ strategie stažení – buď **RAW** anebo **HEADLESS**). Napojuje se na službu *URL Fetcher* skrze volání HTTP requestů. Vrací objekt s informací o staženém obsahu:

- **success** (boolean) – příznak, zda-li byl obsah stažen úspěšně,
- **status** (number) – HTTP status kód odpovědi,
- **data** (string) – stažený obsah (vrací čistě textový řetězec, parsování HTML je implementované v rámci zpracování transformace),
- **headers** (array) – hlavičky odpovědi,
- **error** (Error) – objekt s informací o chybě, pokud nastala.

5.3.7 Transformace dat

Pro transformaci dat je využita služba **TransformService**. Je implementována v souboru `/lib/transformService/transform.ts` a poskytuje funkci `transformHTMLtoJSON`, která přijímá parametry `html` (HTML obsah dokumentu) a `schema` (transformační schéma – kořenový *NodeType*). Uvnitř využívá knihovnu *Cheerio* pro parsování vstupního HTML na DOM strom. Schéma prochází skrze volání transformačních funkcí (`transformNode`) jednotlivých uzlů *NodeType*, které vrací výstupní hodnotu pro daný uzel. Výstupem celé funkce je objekt s následujícími parametry:

- **success** (boolean) – příznak, zda-li byla transformace úspěšná,
- **data** (object) – výstupní data (libovolný typ, v závislosti na transformačním schématu),
- **context** (TransformationContext) – kontext transformace, obsahuje informace o selháních extrakce, které se vyskytly při transformaci.

Protože kód na straně serveru běží v prostředí Node.js, je možné využít transformační služby jak na straně frontendu, tak na straně backendu. Pro frontend je využito v rámci editoru pro vytvoření transformačního schématu. Na straně backendu je pak využito v případě volání veřejného API pro získání výsledku transformace. Výhodou je, že díky jednotné implementaci vrací konzistentní data na obou stranách, jak na straně klienta, tak na straně serveru.

5. IMPLEMENTACE PROTOTYPU

```
import {z} from "zod";

type NodeTypeDefinition<NodeType extends { type: /*...*/ }> = {
  isOfType: (nodeType: any) => nodeType is NodeType,
  // NodeSchemaType je typ sjednocení všech možných typů uzlů,
  // předán jako parametr pro umožnění rekurzivní definice uzlů
  typeSchema: (nodeSchema: z.ZodType<NodeSchemaType>)
    => z.ZodType<NodeType>,
  transformNode: TransformationNodeFunctionType</*...*/>,
  defaultValue: (previousValue?: /*...*/>) => NodeType,
}

type ArrayNodeType = {
  type: "array",
  selector: string,
  items: NodeSchemaType,
};

export const arrayType: NodeTypeDefinition<ArrayNodeType> = {
  isOfType: (nodeType: any) => nodeType?.type === "array",
  typeSchema: (nodeSchema) => z.object({
    selector: z.string(),
    type: z.literal("array"),
    items: nodeSchema,
  }),
  transformNode: (self, contextNode, schema, path, context) => {
    // return [...];
  },
  defaultValue: (previousValue) => {
    // return { type: "array", selector: "", items: ... };
  }
}
```

Výpis kódu 5.3: Definice uzlu pro typ *array*

Oproti návrhu je reprezentace transformačního schématu oddělena od logiky samotné transformace. Transformační schéma je reprezentováno jako složený objekt (v databázi pak uložený jako JSON), kde každý uzel má strukturu definovanou pomocí Zod schématu, zjednodušená ukázka celé definice pro uzel typu *array* je uvedena v kódu 5.3, kde:

- `isOfType` – *type guard*²⁸, který umožňuje typovou kontrolu v rámci Ty-

²⁸<https://www.typescriptlang.org/docs/handbook/2/narrowing.html#using-type-predicates>

peScriptu,

- `typeSchema` – Zod schéma pro kontrolu struktury uzlu,
- `transformNode` – logika samotné transformační funkce,
- `defaultValue` – výchozí hodnota uzlu.

Funkce `transformNode` přijímá parametry `self` (reference na funkci `transformNode`), `contextNode` (DOM element, který odpovídá aktuálnímu uzlu), `schema` (uzel/podstrom transformačního schématu), `path` (cesta k aktuálnímu uzlu uvnitř schématu) a `context` (kontext transformace). Vrací výstupní hodnotu pro daný uzel. Pomocí *contextu* může přidávat chyby v průběhu transformace. (Pro typ *array* např. pokud nenajde žádný uzel se zadaným selectorem.)

Všechny uzly jsou pak spojené dohromady. Výsledné Zod schéma je pak možné použít pro validaci struktury transformačního schématu. zjednodušená ukázka je uvedena v kódu 5.4. Ukázka příkladu objektu, který vybírá titulek a seznam odstavců ze stránky, je na výpisu kódu 5.5.

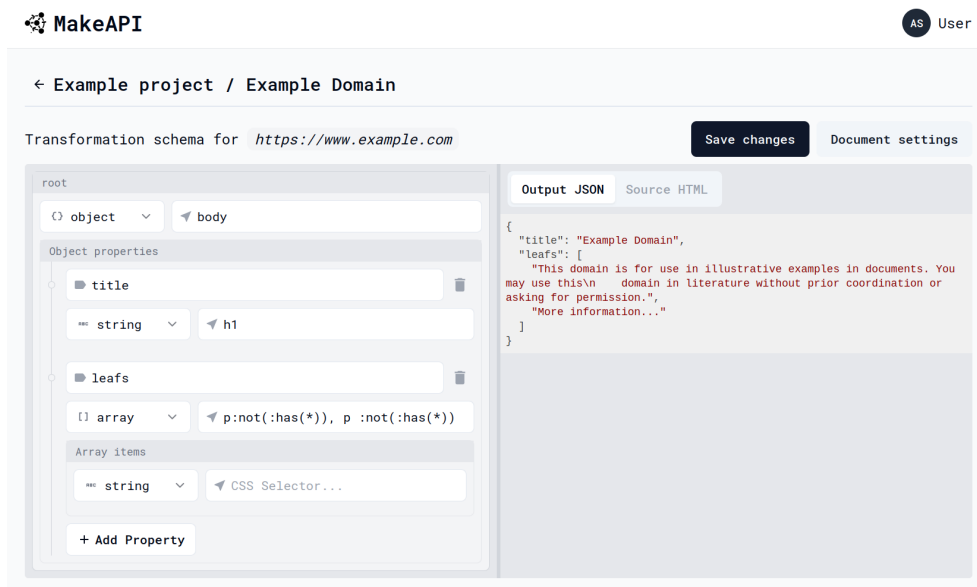
```
type NodeSchemaType = ArrayNodeType | ObjectNodeType | ...;

// Recursive schema must be defined lazily
export const nodeSchema: z.ZodType<NodeSchemaType> = z.lazy(
  () => {
    const s = stringType.typeSchema(nodeSchema);
    const n = numberType.typeSchema(nodeSchema);
    const b = booleanType.typeSchema(nodeSchema);
    const o = objectType.typeSchema(nodeSchema);
    const a = arrayType.typeSchema(nodeSchema);
    const u = nullType.typeSchema(nodeSchema);
    return z.union([s, n, b, o, a, u]);
  });
```

Výpis kódu 5.4: Kombinace všech uzlů dohromady

```
{ type: "object",
  selector: "body",
  properties: [
    { property: "main_title", type: "string", selector: "h1" }
    { property: "text", type: "array", selector: "p",
      items: { type: "string", selector: "" } }
  ]
}
```

Výpis kódu 5.5: Ukázka reprezentace objektu transformačního schématu



Obrázek 5.2: Transformační editor

5.4 Uživatelské rozhraní

Pro implementaci uživatelského rozhraní je využita knihovna *React.js*, pro základní stylování pak knihovna *Tailwind CSS*. V rámci aplikace jsou také využity komponenty z knihovny *shadcn/ui*²⁹. Pro zobrazení ikonky byla využita knihovna *react-icons*³⁰ a *lucide-react*³¹. Pro generování grafických prvků (úvodní grafika, logo, ...) byl využit nástroj *DALL · E*³². Pro zvýraznění syntaxe byla použita knihovna *react-syntax-highlighter*³³

Ukázka UI transformačního editoru je zobrazena na obrázku 5.2. Komponenta je implementována v `components/transformation/EditorWindow.tsx`. Pro ukládání stavu využívá knihovnu *Redux*. Pro zobrazení formuláře editoru jsou využity komponenty *TransformationNode*, které na vstupu přijímají parametry `node` (podstrom transformačního schématu) a `onSchemaUpdate` (callback, který komponenty volají při změně hodnoty).

Editor je rozdělený na dvě okna zobrazená vedle sebe. V levé části je zobrazen formulář pro zadání transformačního schématu. V pravé části je pak

²⁹<https://ui.shadcn.com/>

³⁰<https://react-icons.github.io/react-icons/>

³¹<https://lucide.dev/>

³²<https://openai.com/research/dall-e/>

³³<https://react-syntax-highlighter.github.io/react-syntax-highlighter/>

zobrazen náhled extrahovaných dat ze zadané URL dokumentu. Zároveň je možné zobrazit náhled HTML.

Formulář editoru se skládá z jednotlivých **bloků**, které reprezentují daný uzel ve stromové reprezentaci transformačního schématu. Každý uzel má vždy definovaný *typ* (`string`, `number`, `array`, ...) a některé mají definovaný *selector* (CSS selector, který vybírá požadovaný prvek v DOM), který je vždy relativní vůči rodičovskému selectoru. (Selector daného uzlu tedy vždy vybírá v podstromu DOM určeného selectorem rodičovského uzlu. První uzel, tzv. *root*, odpovídá celému dokumentu, standardně jde o element `<html>...</html>`.)

Celá aplikace je psaná v angličtině. Hlavním důvodem je open-source povaha projektu, kde je snazší využít angličtinu jako primární jazyk. (Umožní to přístup širší skupině vývojářů.) V rámci prototypu jsme neimplementovali překlady do češtiny, avšak v případě potřeby by bylo možné využít například knihovnu *react-i18next*³⁴.

5.5 URL Fetcher

Služba pro stahování dat z webu je přístupná skrze HTTP API endpoint. Jedná se o proxy server, který poskytuje metody pro stažení obsahu ze zadané URL adresy (viz tabulka 5.1).

Metoda	Popis
GET <code>/raw?url=...</code>	Stáhne obsah z URL (bez dalšího zpracování)
GET <code>/html?url=...</code>	Stáhne obsah z URL pomocí headless browseru a vrátí HTML vykreslený v rámci prohlížeče

Tabulka 5.1: URL Fetcher API

Pro stahování dat jsou využity knihovny Axios (pro endpoint `/raw`) a Puppeteer (pro endpoint `/html`). Obě varianty využívají pro cachování Redis databázi. Pro parsování Cache-Control hlaviček využívá knihovnu *cache-control-parser*³⁵. Informace o cachování jsou, vedle samotného obsahu HTTP odpovědi, uloženy do databáze, aby bylo možné následně vypočítat hodnotu `max-age` pro následné odpovědi na API endpoint. V případě, že je obsah cache *fresh*, je vrácena cachovaná odpověď. V opačném případě dojde k novému volání cílové URL.

Pro HTTP requesty odeslané z URL Fetcher vně je přidána hlavička **User-Agent** pro identifikaci služby. (Hodnotu lze upravit v ENV proměnné `USER_AGENT_STRING`.) Zároveň je možné nastavit omezení velikosti staženého

³⁴<https://react.i18next.com/>

³⁵<https://www.npmjs.com/package/cache-control-parser>

5. IMPLEMENTACE PROTOTYPU

obsahu (pro *http* i *raw* requesty zvlášť). V případě, že je velikost staženého obsahu větší než je povoleno, je vrácena chybová odpověď s HTTP kódem 413 – *Payload Too Large*.

Testování

V této kapitole probereme testování aplikace MakeAPI a jejích služeb. Nejprve je popsáno testovací prostředí, které jsme pro testování využili. Následně jsou popsány manuální testy, které jsme provedli v průběhu vývoje. Dále jsou popsány automatizované testy, které jsme vytvořili pro automatizované testování aplikace. V závěru kapitoly pak shrnujeme poznatky a dáváme doporučení pro další vývoj a testování.

6.1 Testovací prostředí

Pro testování aplikace jsme využili nástroj **Docker**, který umožňuje vytvoření izolovaných kontejnerů, ve kterých je možné spouštět aplikace. Tento nástroj je vhodný pro testování, protože umožňuje vytvořit identické prostředí, jako je produkční. Pro nasazení více kontejnerů najednou jsme využili nástroj **Docker Compose**³⁶. Jednotlivé kontejnery jsou definovány jako *služby* v souboru `docker-compose.yml`.

Pro sestavení celé aplikace dohromady jsme vytvořili oddělený git repozitář **make-api-tests**³⁷, který spojuje kód hlavní aplikace MakeAPI a služby URL Fetcher pomocí git submodulů³⁸.

Pro sestavení kontejnerů pro `make-api-fe` a `make-api-be-fetcher` jsou využity vlastní Dockerfile soubory. Pro nastavení proměnných prostředí jsou doplněny soubory `.env.tests.local`³⁹.

Repozitář také obsahuje implementaci **mock serveru** (`mock-app`), který je v rámci testů využit pro testování různých scénářů transformace a přístupu

³⁶<https://docs.docker.com/compose/>

³⁷Dostupný na <https://gitlab.fit.cvut.cz/vanison/d/make-api-tests>

³⁸Více info na <https://git-scm.com/book/en/v2/Git-Tools-Submodules>; v rámci repozitáře jsou využity submoduly `make-api-fe` a `make-api-be-fetcher`

³⁹V rámci jednotlivých submodulů jsou `.env*.local` soubory ignorovány pomocí `.gitignore`, je potřeba je před testováním doplnit ručně do každého zvlášť.

k datům (cache, chybové stavy atp.). Využívá knihovnu Express.js pro zpracování příchozích requestů.

Pro automatizované *end-to-end* testy je využita knihovna **Playwright**⁴⁰. Umožňuje specifikovat scénáře v rámci jednotlivých testů v jazyce TypeScript, které jsou následně spuštěny v prohlížeči. Testy simulují reálné chování uživatele, například klikání na tlačítka, vyplňování formulářů atd.

Pro testování odesílání e-mailů byl využit program **MailDev**⁴¹, který poskytuje SMTP server a webové rozhraní pro zobrazení odeslaných e-mailů.

Struktura repozitáře obsahuje následující soubory a složky:

```

make-api-tests/
├── make-api-fe/ ..... MakeAPI aplikace
│   ├── .env.tests.local
│   └── ...
├── make-api-be-fetcher/ ..... URL Fetcher
│   ├── .env.tests.local
│   └── ...
├── mock-app/ ..... Mock server
│   ├── public/
│   ├── Dockerfile
│   └── server.js
│   └── ...
├── tests/ ..... Adresář pro Playwright testy
│   └── ...
├── docker-compose.yml
├── Dockerfile.make-api-be-fetcher
├── Dockerfile.make-api-fe
├── make-api-fe.init-db.sh
├── README.md
└── ...

```

6.1.1 Nasazení kontejnerů

Pro sestavení a spuštění kontejnerů pomocí nástroje Docker Compose je potřeba spustit následující příkazy:

```

$ docker-compose build
$ docker-compose up -d

```

Po spuštění je potřeba ještě inicializovat databázi (vytvoření tabulek, přidání admin uživatele) pomocí skriptu `make-api-fe.init-db.sh` (požaduje *connection URL* pro připojení k PostgreSQL serveru):

⁴⁰<https://playwright.dev/>

⁴¹<https://maildev.github.io/maildev/>

```
$ ./make-api-fe.init-db.sh
# Enter PostgreSQL connection URL: postgresql://...
```

Jednotlivé služby jsou nakonfigurované tak, aby k nim bylo možné přistupovat i z hostitelského stroje. To je využito např. pro připojení k databázi z vnější sítě (pro inicializaci schématu databáze, či kontrolu uložených dat pomocí DB nástroje). Při produkčním nasazení budou jednotlivé služby dostupné pouze z vnitřní sítě (mezi jednotlivými kontejnery), veřejně přístupná bude pouze aplikace MakeAPI. V rámci vnitřní sítě jsou jednotlivé služby adresovatelné pomocí *hostname*, který odpovídá jménu služby (*service*) v souboru `docker-compose.yml`. Přehled dostupných služeb je zobrazen v tabulce 6.1.

Každá služba poskytuje port (anebo více portů), skrze který je dostupný uvnitř sítě (*vnitřní port*) a skrze který je dostupný z vnější sítě (*vnější port*). Například pokud chce kontejner `make-api-fe` komunikovat se službou URL Fetcher, využije URL: `http://make-api-be-fetcher:4001`. Stejná služba by byla z vnější sítě dostupná skrze URL: `http://localhost:4002`.

Pro zobrazení aplikace MakeAPI v prohlížeči při tomto nasazení lze přistoupit skrze URL: `http://localhost:3000`.

Služba	Service/hostname	Vnější port	Vnitřní port
MakeAPI	<code>make-api-fe</code>	3000	3000
PostgreSQL	<code>make-api-db-postgres</code>	5432	5432
URL Fetcher	<code>make-api-be-fetcher</code>	4002	4001
Redis cache	<code>make-api-be-redis</code>	6379	6379
Mock server	<code>mock-app</code>	3001	8080
MailDev	<code>make-api-maildev</code>	1080, 1025	1080, 1025

Tabulka 6.1: Přehled dostupných služeb/kontejnerů

6.2 Manuální testy

Aplikace byla v průběhu vývoje ručně testována na lokálním stroji. V této sekci popisujeme, jaké testy proběhly manuálním testováním. Část testů byla následně zakomponována do automatizovaných testů, ty následně popíšeme v kapitole 6.3.1 - End-to-end testy.)

6.2.1 Autentizace a autorizace

Ověřili jsme, že k projektům, dokumentům a transformacím lze přistoupit pouze po přihlášení. Uživatel může zobrazit, upravit a smazat pouze své pro-

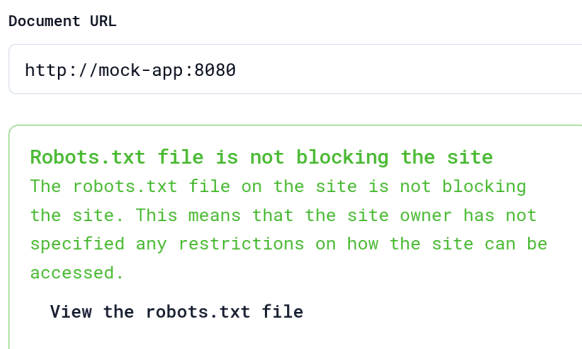
jekty a dokumenty. Může také nastavit přístupové údaje a profilové informace. Pokud je však uživatel navíc v roli **ADMIN**, může přistoupit k aministraci a přehledu všech registrovaných uživatelů a jejich správě. Administrátor může upravit informace ostatních uživatelů a jejich projektů a dokumentů.

K transformacím dokumentů skrze API endpoint lze přistoupit pouze s validním access tokenem. Ověřili jsme, že lze přistoupit pouze k projektu, který je asociovaný s daným tokenem.

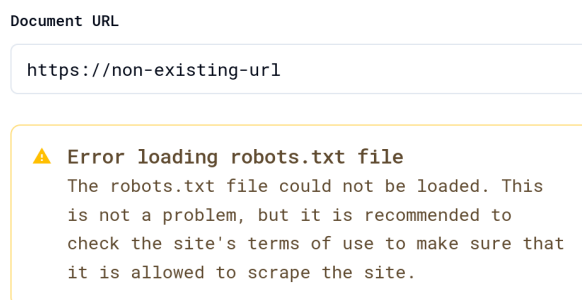
6.2.2 Kontrola robots.txt

V dialogu pro přidání dokumentu (anebo jeho editaci) se načítá informace z `robots.txt` souboru na doméně ze zadané URL. Otestovali jsme tři různé scénáře:

1. **validní přístup** – soubor existuje a povoluje automatické zpracování zadané URL (viz obrázek 6.1),
2. **nenáčtený soubor** – soubor neexistuje, nebo se nepodařilo načíst (viz obrázek 6.2),
3. **zakázaný přístup** – soubor existuje, ale zakazuje přístup k zadané URL (viz obrázek 6.3).



Obrázek 6.1: robots.txt umožňuje přístup ke zdroji



Obrázek 6.2: robots.txt se nepodařilo načíst



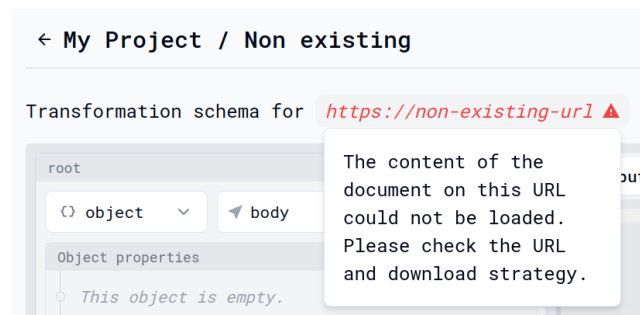
Obrázek 6.3: robots.txt zakazuje přístup ke zdroji

6.2.3 Chyba při načtení dokumentu ze zadané URL

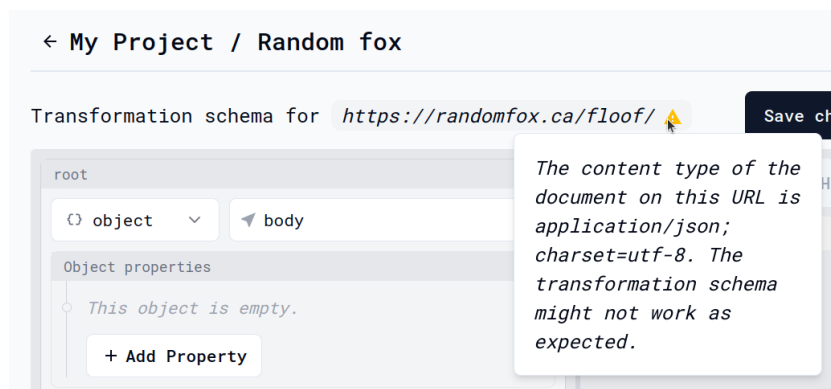
Pokud se dokument nepodaří načíst (například z důvodu, že zadaná adresa je nevalidní, anebo server pro zadanou URL vrátil nějakou chybu, třeba HTTP 500 – Internal Server Error), UI zobrazí informaci v rámci editoru transformačního schématu (viz obrázek 6.4).

Pokud se dokument načte, ale jeho `Content-Type` není `text/html`, může to znamenat, že se jedná o jiný typ dokumentu než HTML, například PDF, JSON, XML atd., a transformace nemusí fungovat správně. V takovém případě je zobrazeno upozornění (viz obrázek 6.5).

6. TESTOVÁNÍ



Obrázek 6.4: Nepodařilo se načíst dokument na zadané URL



Obrázek 6.5: Neočekávaný Content-Type

6.2.4 Stažení obsahu z webu

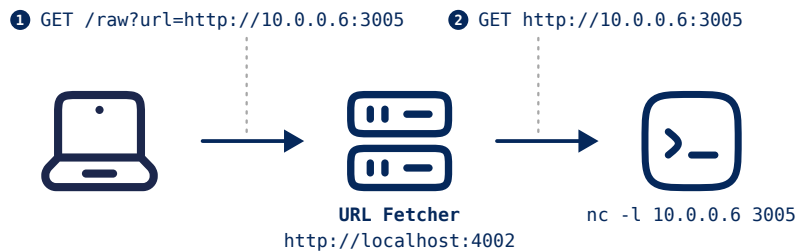
V rámci testování jsme ověřili, že služba URL Fetcher dokáže stáhnout obsah z webu. V této části se zaměřujeme na správné nastavení HTTP hlaviček a chování cache.

6.2.4.1 User-Agent

Služba URL Fetcher odesílá HTTP requesty na web, aby získala obsah dokumentu. V rámci testování jsme využili nástroj `nc` (netcat), který umožňuje naslouchat na zadaném portu a zobrazovat příchozí requesty. Využili jsme ho pro kontrolu, zda služba odesílá správné hlavičky v requestu.

V tomto testu jsme se zaměřili na správně nastavený `User-Agent` pro identifikaci služby, která odesílá request. Ukázka pro `raw` request je na výpisu kódu 6.1 a pro `html` request na výpisu kódu 6.2. Nástroj `nc` je spuštěný na serveru 10.0.0.6 a poslouchá na portu 3005.

Testování probíhalo skrze odesílání requestů na `http://localhost:4002`, na kterém běží služba URL Fetcher.



Obrázek 6.6: Nastavení testu pro ověření hlaviček odeslaných z URL Fetcheru

Z testovaných requestů je vidět, že jsou doplněné i další hlavičky (Accept, Accept-Encoding, ...). Tyto hlavičky doplňuje knihovna Axios (pro *raw* requests) a Puppeteer (pro *html* requests).

```
$ nc -l 10.0.0.6 3005
GET / HTTP/1.1
Accept: application/json, text/plain, */*
User-Agent: MakeApi/1.0
Accept-Encoding: gzip, compress, deflate, br
Host: 10.0.0.6:3005
Connection: keep-alive
```

Výpis kódu 6.1: GET http://localhost:4002/raw?url=http://10.0.0.6:3005

```
$ nc -l 10.0.0.6 3005
GET / HTTP/1.1
Host: 10.0.0.6:3005
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: MakeApi/1.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
      image/avif,image/webp,image/apng,*/*;q=0.8,
      application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

Výpis kódu 6.2: GET http://localhost:4002/html?url=http://10.0.0.6:3005

6.2.4.2 Cache

Pro kontrolu zpracování HTTP requestů s využitím cache jsme využili mock serveru `mock-app`, který pro účely tohoto testu poskytuje následující endpointy:

6. TESTOVÁNÍ

- `/cache/cache-control` – vrací response s hlavičkou `Cache-Control`, která určuje, jak dlouho má být obsah cachován. Pro testování jsme nastavili hodnotu na `public, max-age=10`, která určuje, že obsah je *fresh* po dobu 10 sekund.
- `/cache/etag` – vrací hlavičku `ETag`, která určuje identifikátor obsahu. Na request, který obsahuje hlavičku `If-None-Match` s příslušným tagem, odpoví s HTTP 304 – Not Modified.
- `/cache/last-modified` – vrací hlavičku `Last-Modified` s časem poslední modifikace obsahu. Na request s hlavičkou `If-Modified-Since` s příslušným časem odpoví s HTTP 304 – Not Modified.

Zkoumali jsme chování mock serveru a služby URL Fetcher pomocí výpisu logů. Mock server vypisuje informace o *příchozích requestech* a služba URL Fetcher vypisuje informace o *požadovaných URL* a zda-li byla použita cache (*cache-hit*).

Pro ověření chování jsme v aplikaci MakeAPI vytvořili dokumenty s následujícími URL:

- `http://mock-app:8080/cache/cache-control`
- `http://mock-app:8080/cache/etag`
- `http://mock-app:8080/cache/last-modified`

Pro všechny dokumenty jsme zkoumali *simple fetch* a *headless* varianty requestu. První vede na volání URL Fetcheru na `raw`⁴² a druhý na `html`⁴³ endpoint. Ukázka komunikace pro dokument s URL směřující na mock-app s headless strategií zpracování je zobrazena na obrázku 6.7. Kontejnery na obrázku jsou vzájemně adresovatelné v rámci vnitřní sítě (automaticky vytvořené nástrojem Docker Compose): `make-api-tests_default`.

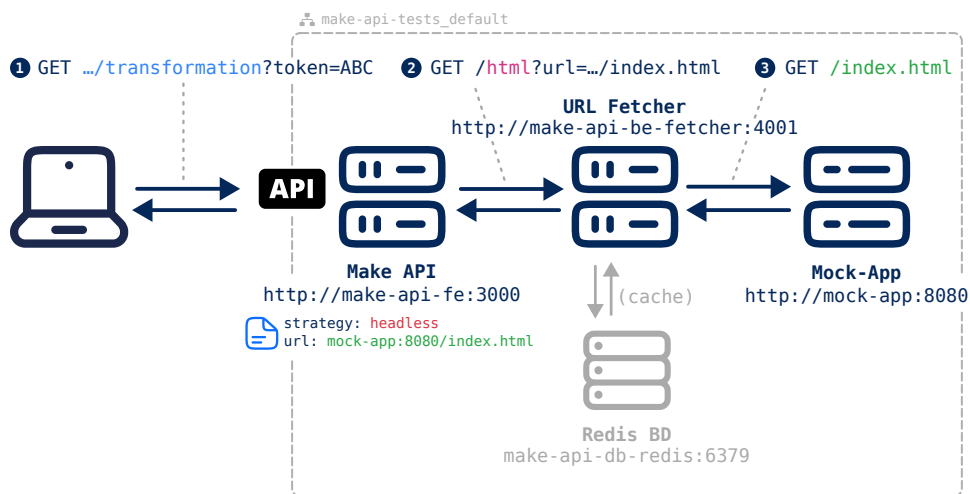
Pro `raw` requesty na `/cache/cache-control` vidíme, že při odeslání prvního requestu je obsah načtený z mock serveru (HTTP 200 – OK), při následném volání jsou načteny data z cache (*cache hit*). Po uplynutí **10 sekund** je následný request opět načten z mock serveru (HTTP 200 – OK). Chování je vidět na výpisu kódu 6.3 pro mock server a 6.4 pro URL Fetcher.

```
[10:00:00]: GET 200 /cache/cache-control
[10:00:11]: GET 200 /cache/cache-control
```

Výpis kódu 6.3: Mock server log znázorňuje, že na server přišly pouze dva requesty ve sledovaném období

⁴²`http://make-api-be-fetcher:4001/raw?url=...`

⁴³`http://make-api-be-fetcher:4001/html?url=...`



Obrázek 6.7: Ukázka komunikace mezi kontejnery v testovacím prostředí

```
[10:00:00] RAW: http://mock-app:8080/cache/cache-control
[10:00:01] RAW: http://mock-app:8080/cache/cache-control (cache)
# ... 10 sekund
[10:00:09] RAW: http://mock-app:8080/cache/cache-control (cache)
[10:00:11] RAW: http://mock-app:8080/cache/cache-control
[10:00:12] RAW: http://mock-app:8080/cache/cache-control (cache)
# ...
```

Výpis kódu 6.4: URL Fetcher log zobrazuje, že většina požadavků byla načtena z cache ve sledovaném období. Nový request byl odeslán na server až po uplynutí 10 sekund

Pro *raw* requesty na `/cache/cache-control` a `/cache/last-modified` nám server nedává informaci o tom, jak dlouho má být obsah cachován. V tomto případě se využije defaultní hodnota nastavená proměnnou prostředí⁴⁴ (v našem testu je hodnota nastavena na 60 sekund). Chování je znázorněné na výpisu kódu 6.5.

⁴⁴Proměnná `URL_CACHE_RAW_CONTENT_DEFAULT_TTL_S`

6. TESTOVÁNÍ

```
[10:09:00] RAW: http://mock-app:8080/cache/etag
[10:09:01] RAW: http://mock-app:8080/cache/etag (cache)
# ... 1 minuta
[10:09:59] RAW: http://mock-app:8080/cache/etag (cache)
[10:10:01] RAW: http://mock-app:8080/cache/etag
[10:10:02] RAW: http://mock-app:8080/cache/etag (cache)
# ...
```

Výpis kódu 6.5: URL Fetcher log pro *raw* requesty, které nepoužívají Cache-Control (zde ukázáno na příkladu pro ETag)

Pro *raw* requesty se využívá knihovna Axios s knihovnou *axios-cache-interceptor*, která ovlivňuje chování cachování pro jednotlivé typy. Z testu bylo zjištěno, že pro ETag a Last-Modified se po uplynutí defaultní doby cache (TTL = 5 minut⁴⁵) vytvoří nový request, avšak bez příslušných hlaviček *If-None-Match* a *If-Modified-Since*. To znamená, že je v obou případech vrácena celá odpověď (HTTP 200 – OK), tj. nevyužijí se *conditional requests* (HTTP 304 – Not Modified). Zde se tedy nachází prostor pro optimalizaci v podobě doplnění příslušných hlaviček do requestu.

Zpracování *html* requestů je komplexnější, protože se využívá headless browser pro načtení a vrácení odpovědi. V tomto případě se využije defaultní hodnota doby pro uložení obsahu do cache⁴⁶ (v našem testu je hodnota nastavena na 5 minut). Z výpisu kódu 6.6 vidíme, že po prvním requestu je obsah načítaný z cache po dobu 5 minut, než dojde k vytvoření nového requestu.

```
[12:30:00] HTML: http://mock-app:8080/cache/last-modified
[12:30:01] HTML: http://mock-app:8080/cache/last-modified (cache)
# ... 1 minuta
[12:34:59] HTML: http://mock-app:8080/cache/last-modified (cache)
[12:35:01] HTML: http://mock-app:8080/cache/last-modified
[12:35:02] HTML: http://mock-app:8080/cache/last-modified (cache)
# ...
```

Výpis kódu 6.6: URL Fetcher log pro *html* requesty

6.2.5 Výpadek Redis DB

Výpadek databáze Redis jsme nasimulovali tak, že jsme vypnuli spuštěný kontejner *make-api-be-redis*. Následně jsme se pokusili přistoupit k dokumentu skrze API endpoint.

Přístup způsobil, že se služba URL Fetcher snažila připojit k Redis databázi, ale připojení selhalo. Tím pádem skončil chybou i celý proces URL

⁴⁵<https://axios-cache-interceptor.js.org/config/request-specifics#cache-ttl>

⁴⁶Proměnná `URL_CACHE_HEADLESS_CONTENT_DEFAULT_TTL_S`

Fetcheru a došlo k restartování celého kontejneru. Navenek se to projevilo jako chyba při přístupu k dokumentu, API endpoint vrátil chybu (HTTP 500 – Internal Server Error) společně s obsahem chyby, který je vidět na výpisu kódu 6.7.

```
{
  "success": false,
  "message": "Failed to fetch HTML for the document: (url)"
}
```

Výpis kódu 6.7: Chyba při transformaci nevalidní URL

6.2.6 Nastavení CORS hlaviček

Pro testování CORS hlaviček v response z API endpointu jsme vytvořili testovací stránku `test-cors.html`, která je dostupná v rámci mock serveru na adrese `http://localhost:3001/test.html`. Stránka obsahuje JavaScriptový kód, který volá API endpoint pro získání transformovaného obsahu dokumentu. Pokud se obsah podaří stáhnout, zobrazí jej vykreslený na stránce. Pokud selže, zobrazí chybu.

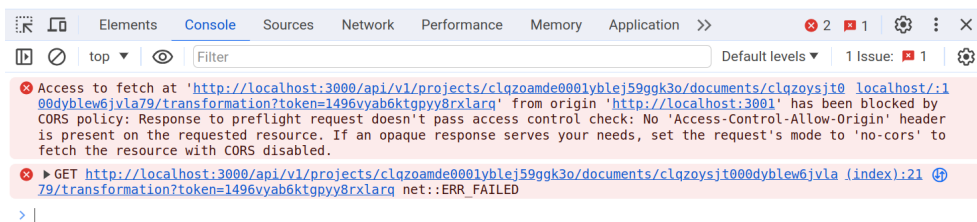
Nejprve jsme otestovali chování bez nastavení *origin* v nastavení dokumentu v aplikaci MakeAPI. V tomto případě se v response nevrací žádné CORS hlavičky. Načtení obsahu z API endpointu je tedy blokováno prohlížečem, ukázka chování je na obrázku 6.8. Na ukázce je vidět, že prohlížeč vypsal chybu do konzole, informující uživatele o chybě při načítání obsahu.

Test of loading API

API Endpoint URL:

API Response:

TypeError: Failed to fetch



Obrázek 6.8: Chyba při načítání obsahu z API endpointu

6. TESTOVÁNÍ

Následně jsme otestovali nastavení dokumentu s nastaveným *origin* na `http://localhost:3001`. V tomto případě došlo ke správnému načtení obsahu a zobrazení v prohlížeči (viz obrázek 6.9).

Test of loading API

API Endpoint URL: `http://localhost:3000/api/v1/projects/clqzoamde000`

API Response:

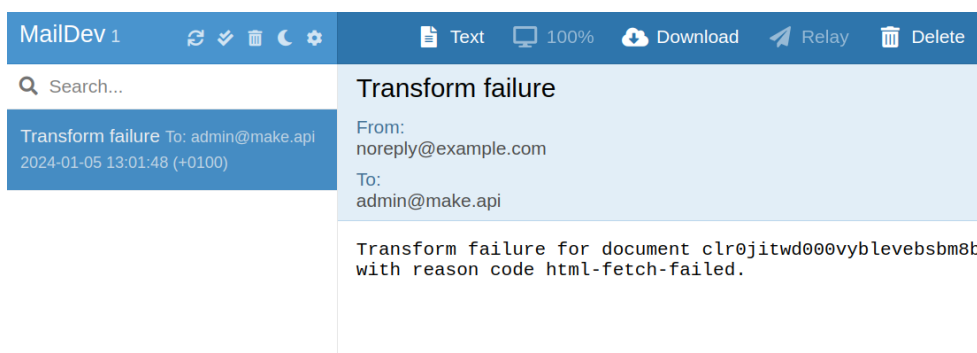
```
{
  "documentName": "JS Content Rendering",
  "sourceUrl": "http://mock-app:8080/samples/js-render.html",
  "data": {
    "has_title": true
  }
}
```

Obrázek 6.9: Úspěšné načtení obsahu z API endpointu

6.2.7 E-mailová notifikace zablokování extrakce

Pro otestování zablokování další extrakce jsme vytvořili dokument s URL `http://mock-app:8080/status/500`, která vždy vrátí zadaný status z URL cesty (zde HTTP 500 – Internal Server Error).

Následně jsme přistoupili k dokumentu skrze API endpoint. Ověřili jsme, že aplikace správně zobrazila chybu extrakce a zaznamenala chybu do databáze. Po uplynutí zadané doby pro zablokování další extrakce (v našem testu nastaveno na 5 sekund) jsme znovu přistoupili k dokumentu. Aplikace opět zobrazila chybu extrakce, ale zároveň nastavila příznak o zablokování další extrakce. Současně odeslala e-mailovou notifikaci na e-mailovou adresu uživatele. Pro samotné odeslání byl použit SMTP server MailDev. Pro kontrolu odeslání jsme přistoupili na webové rozhraní MailDevu, kde byl e-mail vidět (viz obrázek 6.10).



Obrázek 6.10: E-mailová notifikace o zablokování další extrakce

Mezi možné chybové kódy patří následující:

transform-disabled – transformace byla zablokována a nelze ji spustit, dokud nebude odblokována (tj. vyžaduje opravu uživatelem),

schema-invalid – transformační schéma nemá validní strukturu,

html-fetch-failed – nepodařilo se načíst obsah dokumentu z URL Fetcheru (může být dané výpadkem služby, nebo chybou v URL),

transform-failed – transformace selhala chybou a dokument nemohl být zpracován (např. chyba parsování, neexistující typ uzlu, nebo jiná neznámá chyba),

transform-strict-failed – během transformace došlo k chybě, ale je nastavena *striktní transformace*, tj. všechny chyby při transformaci (jako je např. nenalezený HTML element vybraný daným selektorem, případně selektor vrátil více elementů, než kolik je očekáváno, ...) se vyhodnotí jako selhání transformace.

6.3 Automatizované testy

V rámci automatizovaných testů jsme kontrolovali funkčnost a chování aplikace skrze end-to-end testy a unit testy.

6.3.1 End-to-end testy

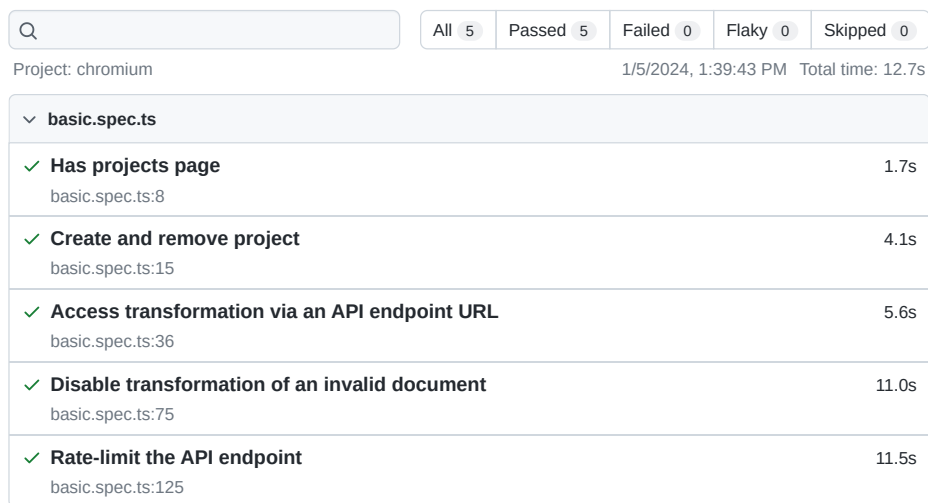
V rámci end-to-end testů jsme otestovali základní funkčnost aplikace pomocí nástroje Playwright. Pro inicializaci testů se vytvoří nový uživatel, skrze kterého se provádí jednotlivé operace. Po dokončení testů je tento uživatel smazán. (Toto odpovídá splnění *UC1: Autentizace a autorizace uživatelů*) Testovali jsme následující scénáře:

Has projects page – Základní test, který kontroluje zobrazení stránky s projekty. Slouží zejména pro kontrolu, že se aplikace načetla a vykreslila správně.

Create and remove project – Testuje vytvoření projektu a přidání dokumentu a jejich následné smazání. Toto odpovídá splnění *UC2: Správa projektů* a *UC3: Správa dokumentů*

Access transformation via an API endpoint URL – Vytvoří nový dokument s jednoduchým transformačním schématem. Testuje zobrazení výsledku transformace skrze API endpoint. Toto odpovídá procesu vytvoření transformace a přístupu skrze API endpoint. Splňuje *UC4.1: Úprava transformačního schématu*, *UC5.1: Vytvoření nového access tokenu* a *UC6.1: Přístup k datům přes API endpoint*.

6. TESTOVÁNÍ



The screenshot shows the Jest test runner interface. At the top, there is a search bar and a summary bar with the following counts: All 5, Passed 5, Failed 0, Flaky 0, and Skipped 0. Below this, the project name 'chromium' and the timestamp '1/5/2024, 1:39:43 PM' are displayed, along with the total time 'Total time: 12.7s'. The main area shows a list of test results for 'basic.spec.ts', all of which passed (indicated by green checkmarks). The tests and their durations are:

Test Name	Duration
Has projects page	1.7s
Create and remove project	4.1s
Access transformation via an API endpoint URL	5.6s
Disable transformation of an invalid document	11.0s
Rate-limit the API endpoint	11.5s

Obrázek 6.11: Výsledek end-to-end testů

Disable transformation of an invalid document – Vytvoří nový dokument s nevalidní URL. Testuje zobrazení chyby a zablokování další extrakce při přístupu skrze API endpoint. Kontroluje chybové stavy (při přístupu HTTP 500, po zablokování pak HTTP 503). V rámci scénáře je také testováno opravení chybové URL a následné obnovení extrakce. Toto odpovídá procesu zablokování transformace při přístupu skrze API. Splňuje *UC6.1: Přístup k datům přes API endpoint* a *UC6.2: Zablokování další extrakce dat*.

Rate-limit the API endpoint – Ověřuje chování rate-limiteru při přístupu skrze API endpoint. Pro projekt nastaví limit na 10 requestů za 5 sekund. Následně provede 10 requestů a ověří, že všechny proběhly úspěšně. Poté provede další request a ověří, že byl zablokován. Po uplynutí 5 sekund provede další request a ověří, že byl úspěšný. Toto odpovídá *UC5.4: Omezení počtu requestů pro access token*.

Výsledek testů je vidět na obrázku 6.11.

6.3.2 Unit testy

Pro ověření funkčnosti transformace dat pomocí transformačního schématu jsme vytvořili unit testy. Testy jsou napsané v jazyce TypeScript a využívají knihovnu Jest. Testy ověřují jednotlivé uzly a správnost jejich transformace.

Výsledek testů je vidět na výpisu kódu 6.8.


```
PASS lib/transformService/__tests__/numberType.test.js
PASS lib/transformService/__tests__/booleanType.test.js
PASS lib/transformService/__tests__/objectType.test.js
PASS lib/transformService/__tests__/transform.test.js
PASS lib/transformService/__tests__/arrayType.test.js
PASS lib/transformService/__tests__/nullType.test.js
PASS lib/transformService/__tests__/selectorSchema.test.js
PASS lib/transformService/__tests__/stringType.test.js
```

```
Test Suites: 8 passed, 8 total
Tests:       46 passed, 46 total
Snapshots:   0 total
Time:        0.671 s, estimated 1 s
Ran all test suites.
```

Výpis kódu 6.8: Výsledek unit testů

6.4 Doporučení na základě testů

Na základě provedených testů jsme identifikovali oblasti, které by bylo vhodné v budoucnu vylepšit:

Při výpadku Redis databáze dojde k **pádu celé služby URL Fetcher**, která přestane fungovat, dokud není databáze znovu dostupná. V takovém případě by bylo možné obejít Redis databázi a dotazovat přímo zadaná URL bez využití cache. Případně by bylo možné využít fallback uložení requestu do paměti, než dojde k obnovení spojení s databází.

Pro samotné využití **cache** se správně aplikují pravidla pro *Cache-Control* hlavičku. Avšak pro *ETag* a *Last-Modified* dochází po vypršení cache v následujících requestech k opětovnému načtení celého obsahu, requesty nevyužívají příslušné hlavičky *If-None-Match* a *If-Modified-Since*. Zde se tedy nachází prostor pro optimalizaci v podobě doplnění příslušných hlaviček do requestu.

6.5 Návrhy na budoucí testování

V rámci testování prototypu jsme otestovali základní funkčnost aplikace. V této sekci popíšeme návrhy na další testování, které by bylo vhodné provést v rámci dalšího vývoje aplikace.

6.5.1 Nasazení v reálném provozu

V rámci testování prototypu jsme využili pouze lokální prostředí. Bylo by vhodné aplikaci také otestovat na **veřejně přístupném nasazeném pro-**

dukčním prostředí, aby bylo možné vyzkoušet chování aplikace v reálném provozu.

6.5.2 Testování na větších datech

V rámci testování prototypu jsme využili pouze několik málo projektů a dokumentů. Pro ověření funkčnosti aplikace by bylo vhodné otestovat **větší množství dokumentů** a **objemnější transformace**, aby bylo možné vyhodnotit, zda-li by bylo potřeba zavést nějaká opatření. (Například omezení na množství uzlů v transformačním schématu, anebo maximální době trvání transformace.)

6.5.3 Chování systému při zátěži a škálování

Simulováním **velkého počtu požadavků** na aplikaci můžeme otestovat chování systému při zátěži, abychom mohli vyhodnotit slabá místa, která by bylo potřeba vhodně škálovat. (Například přidat další instance služby URL Fetcher, anebo Redis databáze.)

Tímto bychom také mohli přesněji určit, jak přísná **pravidla pro rate-limiting** bychom měli nastavit, aby bylo zajištěno, že aplikace bude dostupná pro všechny uživatele i při vysoké zátěži.

6.5.4 Uživatelské testy

Pro ověření použitelnosti by bylo vhodné provést uživatelské testy. Uživatelské testy by mohly odhalit chyby v **uživatelském rozhraní** a **UX**, které by bylo potřeba opravit a vylepšit. Zpětná vazba od uživatelů by nám pomohla se zaměřením pozornosti na problematické části aplikace.

6.5.5 Testování bezpečnosti

V rámci testování prototypu jsme se zaměřili pouze na základní ověření zabezpečení (přístup k projektům po přihlášení, zabezpečení API endpointu pomocí access tokenu). Pro ověření zabezpečení aplikace by bylo vhodné využít komplexnějších testů, které by ověřili bezpečnost (např. proti zranitelnostem XSS, CSRF atd.).

Budoucí práce

V této kapitole provedeme diskuzi a popíšeme náměty a tipy, jakým směrem by bylo možné aplikaci dále vyvíjet a na které aspekty by bylo vhodné se zaměřit v budoucím vývoji.

7.1 Nasazení na produkční prostředí

V aktuálním stavu lze aplikaci spustit a provozovat lokálně v rámci připravených Docker kontejnerů. V budoucnu by bylo vhodné aplikaci nasadit na produkční prostředí, například pomocí nějaké **cloudové služby** (např. AWS, Azure aj.).

Současně by bylo vhodné doplnit podporu pro širší **monitoring přístupů** na web a jeho používání. Tyto informace by bylo možné využít např. pro vhodné nastavení rate-limitů pro přístup k API, anebo pro zjištění slabých míst a jejich optimalizaci. Informace by také mohly vést k lepšímu porozumění používání aplikace uživateli v reálném prostředí.

Důležité je zmínit, že v případě nasazení by bylo potřeba také nastavit jiný **SMTP server** než ten, který jsme využívali v rámci testů, aby bylo možné odesílat e-maily na reálné adresy.

7.2 Rozšíření možností autentizace

S nasazením do produkčního prostředí také souvisí možnosti autentizace. Aplikace současně dovoluje přihlašování pouze pomocí **e-mailu a hesla**, ale použitá knihovna NextAuth.js podporuje i další možnosti, například přihlašování pomocí **OAuth protokolu**. Tímto by bylo možné přidat podporu pro přihlašování pomocí dalších služeb, například Github, Auth0, Google aj., což by bylo vhodnější pro nasazení do produkčního prostředí.

Vhodné by bylo i doplnění dalších způsobů pro ověřování při přístupu k API endpointu. Aktuálně je možné přistupovat k API pouze pomocí access tokenu, ale v budoucnu by bylo vhodné přidat další možnosti, například pomocí OAuth protokolu, nebo JWT tokenu.

7.3 Využití umělé inteligence

Využití umělé inteligence je v současné době relevantní více než kdy dříve. S rapidním nástupem generativních AI přibývá možností, jak tuto technologii využít v praxi.

V rámci práce bylo vytvořeno uživatelské rozhraní pro definici transformačního schématu. Transformace dat je však v současné době prováděna pouze na základě uživatelem definovaného schématu pomocí UI. Bylo by však možné využít umělou inteligenci **pro detekci struktury dat**, na základě které by se automaticky vytvořilo transformační schéma.

Podobný přístup využívají již existující služby (jako je např. Diffbot, nebo ScrapingAnt, více viz kapitola 2 – Rešerše), které poskytují alternativní možnost určení, jaká data chce uživatel z cílové stránky získat (například pomocí textového pole, v rámci kterého uživatel specifikuje, jaká data požaduje). Využití umělé inteligence by mohlo vést k výraznému zjednodušení procesu definice transformačního schématu.

Příklad ukázky využití ChatGPT modelu pro vygenerování struktury transformačního schématu na příkladu je ukázáno na obrázku 7.1. Propojení se samotnou aplikací by pak bylo možné přímo pomocí API, které služba poskytuje.

7.4 Rozšíření možností transformace

V rámci práce bylo vytvořeno uživatelské rozhraní pro definici transformačního schématu. Transformace dat je však v současné době velmi omezená na základní typy (`string`, `number`, `boolean`, `array`, `object` a `null`).

V případě extrakce komplexních a složitých dat by bylo třeba rozšířit možnosti transformace. Pro lepší porozumění této otázce zde uvádíme, jaké jsou aktuální limity našeho řešení:

1. Omezení CSS selektorů

Výběr prvků je řešen pomocí CSS selektorů. Ty však samy o sobě neumí vybrat **atributy elementů**. Například pokud chceme získat `src` atribut z obrázku, nebo `href` atribut z odkazu, aktuální řešení není dostačující. Zároveň je problém, pokud nějaký selektor vybere neexistující element (může se stát například, že daná stránka časem změní svůj obsah a některý prvek z dokumentu vypadne) – v takovém případě dojde k chybě a transformace selže.

MakeAPI Test ▾ 📄

You
Create a transformation schema for menu items on this URL:
<https://www.starbuckscoffee.cz/cz/promo-napoj>

Please extract image URL, title and a description of each drink
< 2 / 2 >

MakeAPI Test
Based on the structure of the Starbucks promotional beverages page, the transformation schema to extract the image URL, title, and description of each drink would be as follows:

```
json 📄 Copy code  
{  
  type: 'array',  
  selector: '.promo-drinks-item',  
  items: {  
    type: 'object',  
    selector: '',  
    properties: [  
      {  
        property: 'image_url',  
        type: 'string',  
        selector: 'img::attr(src)'  
      },  
      {  
        property: 'title',  
        type: 'string',  
        selector: 'h3'  
      },  
      {  
        property: 'description',  
        type: 'string',  
        selector: 'p'  
      }  
    ]  
  }  
}
```

This schema defines an array of objects, each corresponding to a promotional drink item. For each item, it extracts the image URL, title, and description based on the provided CSS selectors:

- ``img::attr(src)`` is used to extract the source URL of the image.
- ``h3`` is used to extract the title of the drink.
- ``p`` is used to extract the description of the drink.

This schema should effectively capture the required details from the webpage.

📄 👍 🗨️ ↻ ⬇️

📄 Message MakeAPI Test... 📄

ChatGPT can make mistakes. Consider checking important information.

Obrázek 7.1: Využití ChatGPT pro definici transformačního schématu

Řešením by bylo rozšířit možnosti výběru prvků, třeba pomocí **XPath**, anebo **rozšířením zápisu CSS selektorů** (Hezké řešení tohoto problému přináší např. aplikace ScrapingBee, která v rámci rozšíření CSS selektorů umožňuje výběr atributů pomocí zápisu „**@attr**“, například „**img@src**“, „**a@href**“, ..., pro výběr pomocí XPath pak používá zápis s prefixem „/“, např. „//h1[@id=**title**]“, ... [42]).

2. Rozšíření typů uzlů

Jednotlivé základní typy jsou velmi primitivní a samy o sobě neřeší další sémantiku dat. Například pokud chceme získat **data s konkrétním významem** (jako je datum, URL, e-mail, cena aj.), bylo by vhodné doplnit možnost další parametrizace – například pomocí regulárních výrazů, nebo jiných metod.

Zajímavou možností by bylo například přidání **vlastní funkce** pro zpracování uzlu, která by se vykonala nad vybranými daty. Tím by bylo možné provádět komplexnější operace, jako je např. převod dat do jiného formátu, matematické operace aj.

3. Dynamické typy

V rámci transformace je možné definovat pouze statické typy, tzn. že pokud definujeme *objekt*, musí mít vždy **předem daný seznam parametrů**, které se nemění. Pokud bychom však chtěli pro nějaký prvek vybrat data podmíněně (třeba pouze pokud je daný prvek přítomen na stránce), bylo by nutné funkcionalitu rozšířit o dynamické typy.

Můžeme si to představit na příkladu stránky se seznamem produktů, který má definované různé parametry, např. výrobce, model, cena, ..., některé ale mohou mít navíc další atributy, jako je např. sleva, nebo informace o nedostupnosti zboží.

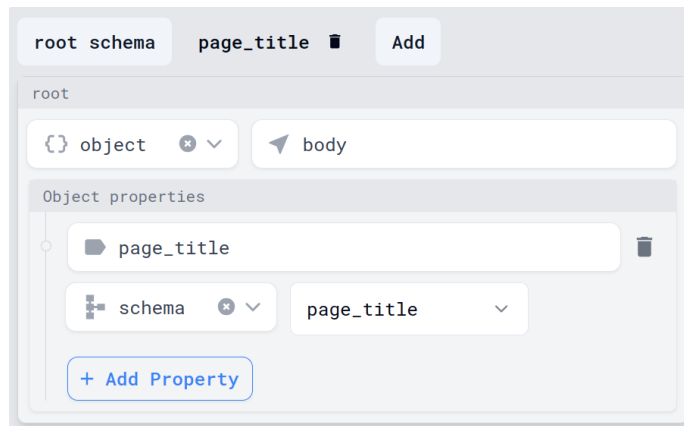
4. Podmíněné uzly

Pokud nějaký uzel přistupuje k datům pomocí selektoru, který adresuje neexistující prvek, anebo jich je více, než kolik uzel předpokládá, dojde k chybě transformace. V takovém případě by bylo vhodné mít možnost definovat podmíněné uzly, které by se v případě, že by nebylo možné vybrat data, vynechaly.

Možné by bylo také doplnit komplexnější podmínky pro výběr dat, např. kontrola data (větší/menší než dnešní datum), kontrola ceny (požadovaný rozsah v Kč) aj.

5. Šablony a rekurzivní schémata

Současně se transformační schéma definuje v rámci jednoho formuláře, který obsahuje celou definici schématu najednou. Pro komplexnější transformace by tato definice mohla rychle narůst a být nepřehledná z důvodu velkého množství formulářových prvků.



Obrázek 7.2: Příklad možného rozšíření o šablony/podschémata

Pro lepší organizaci by bylo možné doplnit možnost přidávat **šablony/podschémata**, která by bylo možné využít ve formuláři jako zástupný prvek. Tím by se zjednodušila a zpřehlednila definice transformačního schématu. (Ukázka možného rozšíření je na obrázku 7.2.)

7.5 Interakce se stránkou

Aplikace umožňuje zpracování dat ze stránky vykreslené pomocí JavaScriptu v headless browseru. V současné době je však možné pouze načíst stránku a získat její HTML obsah. V budoucnu by bylo vhodné rozšířit možnosti interakce se stránkou. Například o možnost kliknutí na tlačítko, vyplnění formuláře aj.

Tuto funkcionalitu v nějaké podobě nabízí většina zkoumaných nástrojů v rešerši. Ať už pomocí klikání na vykreslené stránce v rámci UI, definováním seznamu operací v JSON formátu, anebo pomocí konkrétního JS snippetu pro vykonání operací na stránce.

Přidání těchto operací by mohlo být snadným rozšířením, neboť naše aplikace již využívá headless browser pro načítání stránky. V základním režimu by stačila možnost přidat **časový interval, jak dlouho se má počkat po vykreslení stránky** (pokud je např. potřeba počkat na dokončení asynchronních JavaScriptových operací, jako je načtení externího obsahu a následné vykreslení). Případě umožnit zadat **selector prvku, který musí být vykreslený na stránce**, než bude možné dále pokračovat v transformaci. (Tento přístup využívá např. aplikace ScrapingBee.)

7.6 Crawling

Speciálním typem interakcí je navigování a procházení dalších stránek, které jsou odkazovány v aktuálně zpracovávaném dokumentu, tzv. crawling.

Aktuálně lze aplikaci použít pouze pro zpracování jedné stránky současně. V budoucnu by bylo možné přidat funkcionalitu **zpracování více stránek najednou**, kdyby aplikace poskytla možnost navigování a zpracování odkazů na další stránky, anebo skrze definování seznamu vstupních URL. (Tuto možnost poskytuje například aplikace ParseHub nebo Octoparse.)

7.7 Rozšíření uživatelského rozhraní

Jedním z velkých nedostatků aktuální podoby zadávání transformačního schématu je, že je **nutné znát strukturu HTML dokumentu**, ze kterého chceme data získat. Ačkoliv aplikace zobrazuje i HTML obsah stránky, je komplikované v něm hledat potřebné informace, neboť jde pouze o textovou reprezentaci.

V rámci vývoje jsme se okrajově zabývali i možnostmi **vykreslení stránky přímo v rámci UI** (pomocí HTML prvku `iframe`, viz ukázka na obrázku 7.3) tak, aby uživatel mohl vybírat prvky pomocí klikání na cílové stránce.



Obrázek 7.3: Příklad interaktivního výběru prvků pomocí kurzoru

Tato otázka však přináší další problémy, např. kvůli **CORS politice** prohlížečů, která neumožňuje načítat obsah z jiných domén. V takovém případě by bylo možné využít náš **proxy server** pro načtení obsahu, avšak zde přichází další komplikace, neboť načtený HTML dokument může obsahovat odkazy na další zdroje (např. CSS, JavaScript, obrázky, ...).

Tyto odkazy mohou být *relativní*, pak je problém s tím, že odkazují na špatnou doménu (neboť ta by byla změněna kvůli načtení obsahu pomocí proxy serveru), nebo *absolutní*, pak je opět problém s CORS politikou.

Výsledkem pak je **neúplné načtení obsahu stránky**⁴⁷, protože některé zdroje zkrátka není možné načíst. Tento problém by se dal obejít například pomocí nahrazování regulárním výrazem (všude, kde je nějaký link, by došlo

⁴⁷Neúplné načtení ve smyslu zobrazení pouze obsahu HTML dokumentu s použitím defaultních stylů prohlížeče. Načtení CSS souborů nemusí fungovat správně a zobrazená stránka tak vizuálně bude odlišná.

k nahrazení URL s použitím naší proxy domény), avšak toto řešení není úplně ideální, neboť by mohlo dojít k nekonzistentnímu chování stránky.

Jedná se o velmi komplexní problém, pro který neexistuje jednoduché řešení.

Variantou by však mohlo být využití jiného přístupu pro výběr prvků na stránce. Jelikož načtení skrze `iframe` je velmi omezené (zejména z bezpečnostních důvodů a CORS politiky, jak bylo zmíněno), bylo by možné například **vytvořit plugin pro prohlížeč**, který tato omezení dokáže obejít a umožňuje spouštět JavaScriptový kód na aktuálně otevřené stránce.

Toho by se dalo využít pro výběr prvků na stránce pomocí klikání v otevřeném okně s cílovou stránkou, přičemž plugin by zaznamenával vybrané prvky a pomohl by zjednodušenému vytvoření transformačního schématu.

7.8 Rozšíření možností exportu a validace

Jediným výstupním formátem je v současné době **JSON**. V budoucnu by bylo vhodné rozšířit možnosti exportu. Například o možnost exportu do XML, CSV aj.

Struktura definice transformačního schématu se vágně podobá struktuře, kterou využívá **JSON Schema**⁴⁸ pro definici schématu JSON dokumentů. To by mohlo vést ke snadnému rozšíření aplikace o zavedení exportu JSON Schema pro výstupní data transformace. Koncovému uživateli by to mohlo pomoci při validaci výstupních dat v rámci integrace do jiných systémů.

⁴⁸<https://json-schema.org/specification>

Závěr

V rámci této práce jsme vytvořili prototyp webové aplikace **MakeAPI**⁴⁹, která je zaměřena na extrakci a transformaci dat z jednoduchých HTML stránek. Aplikace poskytuje webové rozhraní pro definici transformačního schématu. K extrahovaným datům následně umožňuje přistoupit skrze API endpoint.

Analyzovali jsme existující technologie a provedli **rešerši** dostupných nástrojů pro scraping a transformaci dat z webu, kde jsme srovnávali vybrané parametry (jako je cena, způsob extrakce dat a její konfigurace, podporované formáty aj.).

Následně jsme provedli **analýzu softwarových požadavků** a modelování nejdůležitějších **procesů aplikace** (vytvoření transformace a přístup skrze API, zablokování transformace). Na základě těchto poznatků jsme navrhli architekturu aplikace a její jednotlivé komponenty. Hlavní komponentou je *TransformService*, což je služba, která zprostředkovává samotnou transformaci dat.

Transformace je definována pomocí *transformačního schématu*. Využívá se rekurzivní datové struktury⁵⁰, která prochází vstupní HTML dokument a na základě definovaných pravidel vytváří výstupní JSON dokument.

V rámci implementace jsme vytvořili **funkční prototyp aplikace** s využitím Next.js frameworku. Pro stahování dat z webu jsme vytvořili samostatnou službu *URL Fetcher*, která umožňuje stahovat data ze zadané URL. Umožňuje také načtení dat v rámci *headless* browseru, což se hodí pro webové stránky, které jsou vykreslené pomocí JavaScriptu. Aplikace poskytuje jednoduché UI pro definici transformačního schématu.

Aplikaci jsme **testovali pouze na lokálním prostředí**. Pro konfiguraci a spuštění testovacího prostředí jsme využili nástroj *Docker Compose*, který

⁴⁹Zdrojové kódy jsou k dispozici v git repozitáři na adrese <https://gitlab.fit.cvut.cz/vanisond/make-api-fe> a <https://gitlab.fit.cvut.cz/vanisond/make-api-be-fetcher/>

⁵⁰Jedná se o strom, kde jednotlivé uzly představují prvek, který má být na výstupu – jedná se o typy `string`, `number`, `boolean`, `array`, `object` a `null`

nám umožňuje spustit aplikaci v kontejnerech.⁵¹ Testování probíhalo z části manuálně (chování cache, nastavení CORS, načtení informací z robots.txt, ...), z části automaticky pomocí testů (end-to-end testy a unit testy).

V závěru práce jsme provedli diskuzi a zmínili možná rozšíření aplikace. V rámci **budoucího vývoje** by bylo vhodné aplikaci nasadit na produkční prostředí a provést dodatečné testy, které by jednak ověřily její funkčnost v reálném prostředí, druhá by pomohly odhalit slabá místa a nastavit přesnější omezení rate-limiteru. Nastínili jsme i další možná rozšíření, např. přidání dalších autentizačních metod, rozšíření transformace o podmíněné typy, šablony/podschémata, či doplnění selectorů o možnost výběru atributů. Vhodné by bylo zaměřit se také na způsob zadávání transformace, např. skrze doplnění vykreslení stránky v rámci UI a následný výběr prvků pomocí kurzoru, anebo využití umělé inteligence pro automatickou detekci a vytvoření struktury transformačního schématu.

⁵¹Celé nastavení a konfigurace testovacího prostředí je dostupná v samostatném repozitáři na adrese <https://gitlab.fit.cvut.cz/vanisond/make-api-tests>

Bibliografie

1. KUCHARŮ, Jaroslav; DOJČINOVSKI, Milan. *Web Data Mining - Lecture 1: Introduction and Course Overview* [online]. 2023. [cit. 2023-12-09]. Dostupné z: <https://courses.fit.cvut.cz/NI-DDW/lectures/files/lecture1-1p.pdf>. Slide 5.
2. VOSSEN, Gottfried; HAGEMANN, Stephan. The client/server principle. In: *Unleashing Web 2.0: From Concepts to Creativity*. Morgan Kaufmann, 2007/2010. ISBN 0123740347;9780123740342.
3. MOZILLA DEVELOPER NETWORK (MDN). *An overview of HTTP* [online]. 2023. [cit. 2023-12-09]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>.
4. MOZILLA DEVELOPER NETWORK (MDN). *HTML: HyperText Markup Language* [online]. 2023. [cit. 2023-12-09]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTML>.
5. WHATWG. *4.1. Introduction to "The DOM"* [online]. 2023. [cit. 2023-12-09]. Dostupné z: <https://dom.spec.whatwg.org/#introduction-to-the-dom>.
6. BERNERS-LEE, Tim; FIELDING, R.; MASINTER, L. *Uniform Resource Identifier (URI): Generic Syntax* [online]. 2005. [cit. 2023-12-09]. Dostupné z: <https://datatracker.ietf.org/doc/html/rfc3986>.
7. SAINT-ANDRE, P.; CROCKER, D.; NOTTINGHAM, M. *Deprecating the "X-"Prefix and Similar Constructs in Application Protocols* [online]. 2012. [cit. 2023-12-09]. Dostupné z: <https://datatracker.ietf.org/doc/html/rfc6648>.
8. MOZILLA DEVELOPER NETWORK (MDN). *HTTP request methods* [online]. 2023. [cit. 2024-01-07]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>.

9. MOZILLA DEVELOPER NETWORK (MDN). *HTTP response status codes* [online]. 2023. [cit. 2023-12-11]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>.
10. MOZILLA DEVELOPER NETWORK (MDN). *Proxy servers and tunneling* [online]. 2023. [cit. 2023-12-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Proxy_servers_and_tunneling.
11. MOZILLA DEVELOPER NETWORK (MDN). *HTTP caching* [online]. 2023. [cit. 2023-12-11]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching>.
12. MOZILLA DEVELOPER NETWORK (MDN). *HTTP conditional requests* [online]. 2023. [cit. 2024-01-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Conditional_requests.
13. MOZILLA DEVELOPER NETWORK (MDN). *Cache-Control* [online]. 2023. [cit. 2023-12-11]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control>.
14. CLOUDFLARE. *What is rate limiting? | Rate limiting and bots* [online]. 2023. [cit. 2023-12-11]. Dostupné z: <https://www.cloudflare.com/learning/bots/what-is-rate-limiting/>.
15. MOZILLA DEVELOPER NETWORK (MDN). *Cross-Origin Resource Sharing (CORS)* [online]. 2023. [cit. 2023-12-11]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>.
16. LIU, Bing. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Berlin Heidelberg New York: Springer Berlin Heidelberg New York, 2007. ISBN 978-3-540-37881-5. Corrected 2nd printing 2008.
17. M. KOSTER AND G. ILLYES AND H. ZELLER AND L. SASSMAN. *RFC 9309 Robots Exclusion Protocol* [online]. 2022. [cit. 2023-12-12]. Dostupné z: <https://www.rfc-editor.org/rfc/rfc9309.html>.
18. KUCHARŮ, Jaroslav; DOJČINOVSKI, Milan. *Web Data Mining - Lecture 3: Data Access and Acquisition Methods 2* [online]. 2023. [cit. 2023-12-12]. Dostupné z: <https://courses.fit.cvut.cz/NI-DDW/lectures/files/lecture3-1p.pdf>. Slide 19—20.
19. SCRAPINGBEE. *ScrapingBee, the best web scraping API* [soft.]. 2023. [cit. 2023-12-13]. Dostupné z: <https://www.scrapingbee.com/>.
20. *Apify* [soft.]. 2023. [cit. 2023-12-14]. Dostupné z: <https://apify.com>.
21. APIFY. *What's an actor? - Actors | Academy* [online]. 2023. [cit. 2023-12-14]. Dostupné z: <https://docs.apify.com/academy/getting-started/actors#what-is-an-actor>.
22. APIFY. *Web Scraper* [online]. 2023. [cit. 2023-12-14]. Dostupné z: <https://apify.com/apify/web-scraper>.

23. *Diffbot* [soft.]. 2023. [cit. 2023-12-14]. Dostupné z: <https://www.diffbot.com/>.
24. *Import.io* [soft.]. 2023. [cit. 2023-12-14]. Dostupné z: <https://www.import.io/>.
25. IMPORT.IO. *Import.io Getting Started May 2017* [video]. 2017. [cit. 2023-12-14]. Dostupné z: <https://www.youtube.com/watch?v=ODyaE96ePNE>.
26. *Octoparse* [soft.]. 2023. [cit. 2023-12-14]. Dostupné z: <https://www.octoparse.com/>.
27. KRISH NAIK. *Octoparse- Easy Web Scraping Tool Without Any Coding Knowledge* [video]. 2021. [cit. 2023-12-14]. Dostupné z: <https://www.youtube.com/watch?v=ODyaE96ePNE>.
28. *ParseHub* [soft.]. 2023. [cit. 2023-12-14]. Dostupné z: <https://www.parsehub.com/>.
29. SCRAPINGHUB. *Portia* [soft.]. 2023. [cit. 2023-12-14]. Dostupné z: <https://github.com/scrapinghub/portia>.
30. MINIRPA. *NoCoding Data Scraper* [soft.]. 2021. [cit. 2023-12-14]. Dostupné z: <https://www.minirpa.net/>.
31. GRAPH, Web. *Web Scraper* [soft.]. 2023. [cit. 2023-12-14]. Dostupné z: <https://www.webscraper.io/>.
32. HANGZHOU DUOSUAN TECHNOLOGY CO., Ltd. *ScrapeStorm* [soft.]. 2020. [cit. 2023-12-14]. Dostupné z: <https://www.scrapestorm.com/>.
33. DATAANT. *ScrapingAnt* [soft.]. 2023. [cit. 2023-12-14]. Dostupné z: <https://scrapingant.com/>.
34. SCRAPERAPI. *ScraperAPI* [soft.]. 2023. [cit. 2023-12-14]. Dostupné z: <https://www.scraperaapi.com/>.
35. SCRAPENINJA. *ScrapeNinja* [soft.]. 2023. [cit. 2023-12-14]. Dostupné z: <https://scrapeninja.net/>.
36. VERCEL. *Next.js Docs - Introduction* [online]. 2023. [cit. 2023-12-18]. Dostupné z: <https://nextjs.org/docs>.
37. LTD., Redis. *Redis* [soft.]. 2023. [cit. 2023-12-18]. Dostupné z: <https://redis.io/>.
38. REFACTORING GURU. *Composite* [online]. 2023. [cit. 2023-12-19]. Dostupné z: <https://refactoring.guru/design-patterns/composite>.
39. KAURA, Patrik. *The Fundamentals of Rate Limiting: How It Works and Why You Need It* [online]. 2023. [cit. 2023-12-19]. Dostupné z: <https://medium.com/@patrikkaura/the-fundamentals-of-rate-limiting-how-it-works-and-why-you-need-it-fd86d39e358d>.
40. REINMAN, Andris. *Nodemailer* [soft.]. 2023. [cit. 2023-12-27]. Dostupné z: <https://nodemailer.com/>.

BIBLIOGRAFIE

41. PRISMA DATA, Inc. *Prisma* [soft.]. 2023. [cit. 2023-12-18]. Dostupné z: <https://www.prisma.io/>.
42. SCRAPINGBEE. *ScrapingBee Docs: Extract data with CSS or XPATH selectors* [online]. 2023. [cit. 2023-01-09]. Dostupné z: https://www.scrapingbee.com/documentation/data-extraction/#selector_type.

Seznam použitých zkratk

AI	Artificial Intelligence
API	Application Programming Interface
BE	Backend
CDN	Content Delivery Network
CORS	Cross-Origin Resource Sharing
CSRF	Cross-Site Request Forgery
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
DB	Database
DNS	Domain Name System
DOM	Document Object Model
FE	Frontend
FR	Functional Requirement
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JS	JavaScript
JSON	JavaScript Object Notation

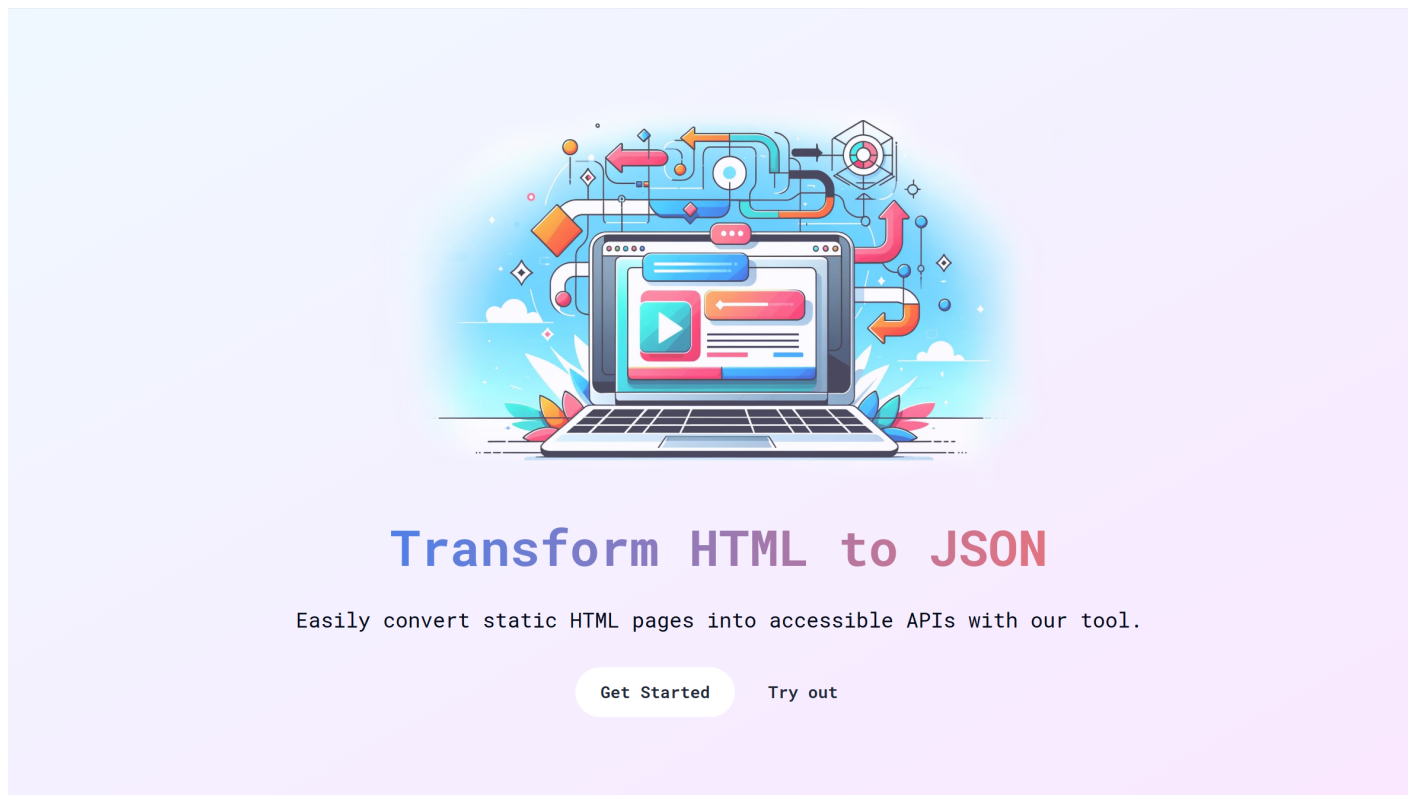
A. SEZNAM POUŽITÝCH ZKRATEK

JWT	JSON Web Token
NPM	Node.js Package Manager
NR	Non-Functional Requirement
ORM	Object-Relational Mapping
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
TS	TypeScript
TTL	Time To Live
UC	Use Case
UDP	User Datagram Protocol
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UX	User Experience
XML	Extensible Markup Language

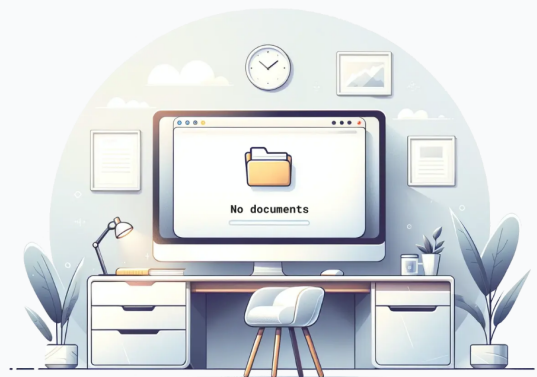
Ukázky UI aplikace

V této příloze doplňujeme ukázky UI aplikace MakeAPI:

B.1: Úvodní stránka	104
B.2: Stránka projektu	105
B.3: Transformační editor	106
B.4: Formulář vytvoření dokumentu v projektu	107
B.5: Formulář pro přidání access-tokenu	108
B.6: Detail profilu uživatele	109
B.7: Možnost smazání účtu	109
B.8: Administrace uživatelů	110
B.9: Nastavení rate-limitů	110



Obrázek B.1: Úvodní stránka



There are no documents in this project yet.

Add document

Obrázek B.2: Stránka projektu

MakeAPI AS User

← Example project / Example Domain

Transformation schema for `https://www.example.com` Save changes Document settings

root

object ▾ body

Object properties

- title 🗑️
string ▾ h1
- leafs 🗑️
array ▾ p:not(:has(*)), p :not(:has(*))

Array items

- string ▾ CSS Selector...

+ Add Property

Output JSON Source HTML

```
{
  "title": "Example Domain",
  "leafs": [
    "This domain is for use in illustrative examples in documents. You
    may use this\n  domain in literature without prior coordination or
    asking for permission.",
    "More information..."
  ]
}
```

Obrázek B.3: Transformační editor

Add document ×

Document URL

Document name

Advanced options ^

Download strategy

Simple fetch Headless

The document will be downloaded using a headless browser. This is a slower option, but it will work for sites that require JavaScript to render the content.

Strict transform
If enabled, the API transformation endpoint will be disabled if some errors are encountered while transforming the document.

Cache control max-age

Specify the max-age value for the Cache-Control header. This will tell the browser how long it can cache the document before it needs to be re-downloaded.


Add document

Obrázek B.4: Formulář vytvoření dokumentu v projektu

Create access token ×

Create a new access token for this project.

Access Token

 Please copy this token and store it somewhere safe. You won't be able to see it again.

Description

Expiration date

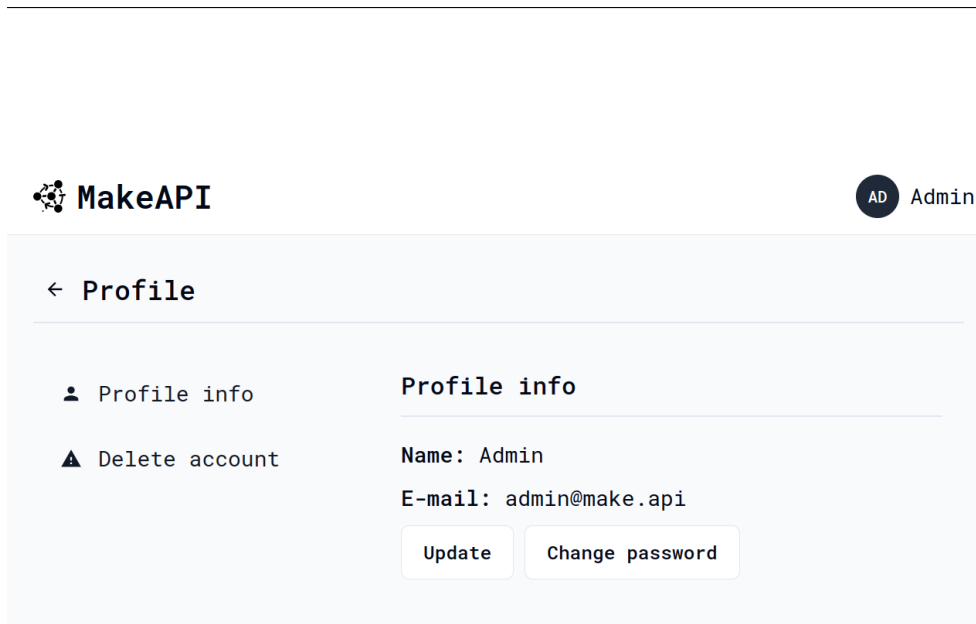
Expires in 12 months
Leave empty for no expiration date.

Advanced options ^

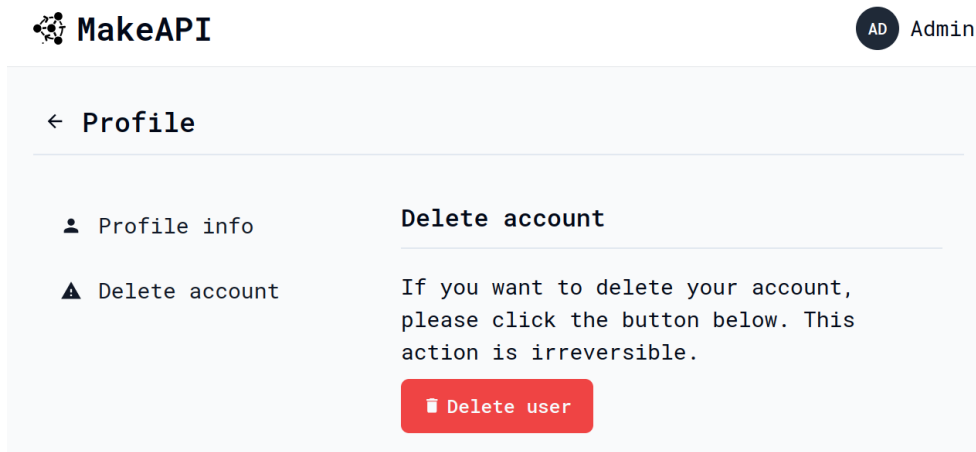
Allowed origins

CORS origins that are allowed to use this token. One origin per line.

Obrázek B.5: Formulář pro přidání access-tokenu

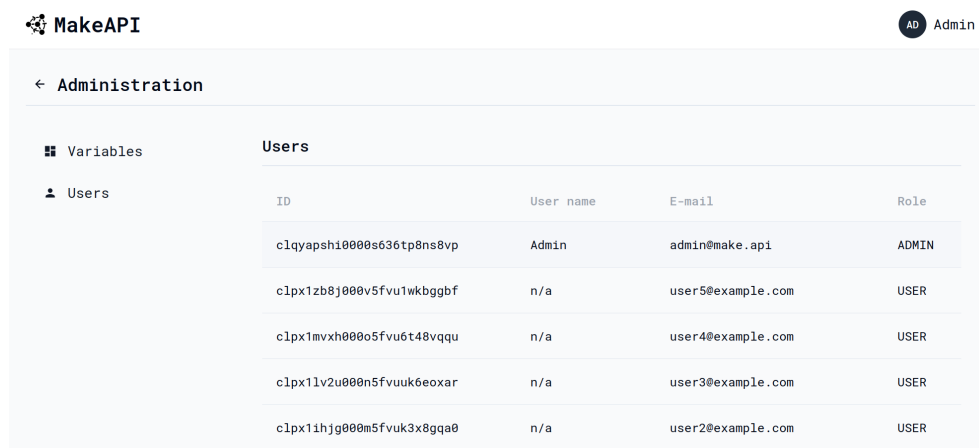


Obrázek B.6: Detail profilu uživatele

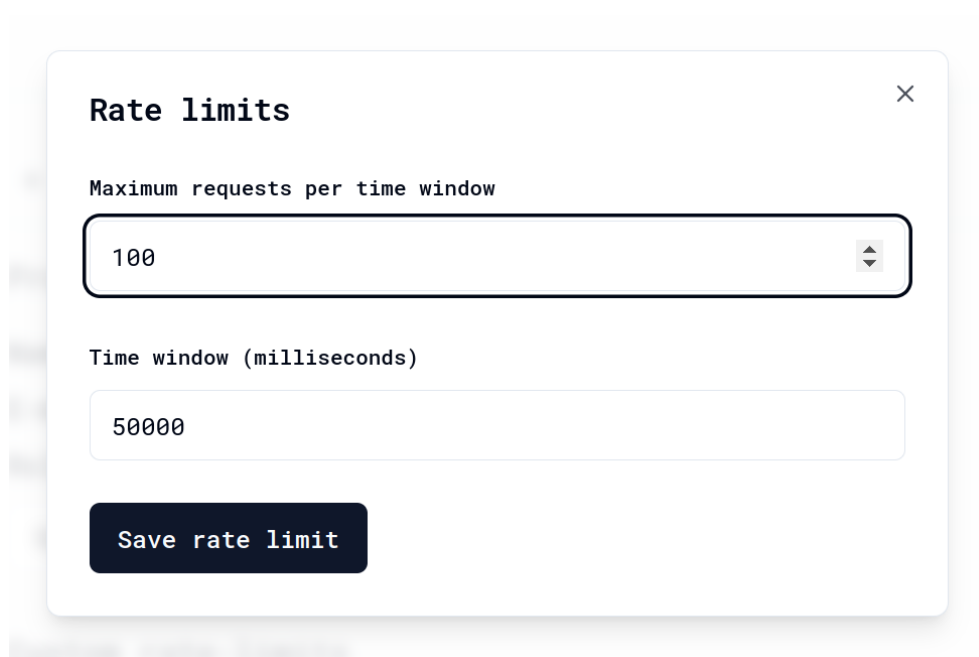


Obrázek B.7: Možnost smazání účtu

B. UKÁZKY UI APLIKACE



Obrázek B.8: Administrace uživatelů



Obrázek B.9: Nastavení rate-limitů

Obsah přiloženého CD

/	
	readme.txt.....stručný popis obsahu CD
	src
	impl.....zdrojové kódy implementace
	make-api-be-fetcher.....služba BE Fetcher
	make-api-fe..... hlavní aplikace (UI, API endpoint)
	make-api-tests.....testy (end-to-end)
	thesis..... zdrojová forma práce ve formátu \LaTeX
	text..... text práce
	thesis.pdf..... text práce ve formátu PDF